64'er Sonderheft Assembler, Programmiersprachen

SONDERHEFT 12 Lit. 12000/hfl. 18,-/dkr. 68,- ÖS 100,-/sfr. 14,- DM 14,-

Markt&Technik

64'er

ASSEMBLER
PASCAL
COMAL
FORTH
PILOT
PROLOG

Assembler

- ★ Sortierprogramme für höchste Ansprüche
- ★ Superschnelle Grafikroutinen
- ★ Maschinensprachemonitor für Profis
- ★ Nützliche Programmodule zum Abtippen

Grundlagen

- ★ Programmieren in Pascal
- ★ dBase II in der Praxis
- ★ Grafik in Maschinensprache
- ★ Das kann der Z80-Prozessor

Programmiersprachen

- ★ Fantastisches Turbo-Pascal
- ★ Preiswerte CP/M-Sprachen
- ★ Test: Die besten Basic-Compiler

CP/M

Alle Programme auch auf Diskette erhältlich



Fremdsprachen für C 64 und C 128



Wenn wir Menschen es doch so leicht hätten: Jemand würde sagen, er möchte jetzt die oder die Sprache mit Ihnen sprechen, gäbe Ihnen eine Pille und kurze Zeit später könnten Sie sich fließend in Japanisch unterhalten. Ein Wunschtraum? Sicherlich. Computer haben es da schon leichter. Diskette rein, Programm laden und schon kann die Unterhaltung beginnen. Zugegeben, sehr einseitig, und immer wieder sind wir Menschen die Schwachstelle, weil wir diese Sprache erst lernen müssen. Gegenüber dem Erlernen einer »menschlichen« Sprache in der Schule gibt es jedoch einige große Vorteile: Computersprachen sind wesentlich einfacher (da künstlich), und das Lernen dieser Sprache macht sehr viel mehr Spaß. Warum eigentlich? Vielleicht, weil man schneller Erfolgserlebnisse hat. Man sieht seine Kenntnisse schneller wachsen, erhält sofort die Bestätigung, ob etwas verkehrt oder richtig war, weil der »Gesprächspartner« zwar pingelig, aber unendlich geduldig ist.

Der Faszination, eine neue, bessere und schnellere Sprache zu lernen, kann man sich kaum entziehen. Allerdings gibt es eine ganze Reihe von Programmen für den C 64 und den C 128, die auch nicht gerade kostenlos sind. Eine Auswahl sollte daher schon getroffen werden. Aus diesem Grund haben wir in diesem Sonderheft versucht, Ihnen soviel wie möglich Informationen zu liefern. Welche Programmiersprache die richtige ist, hängt hauptsächlich vom Einsatzgebiet ab. Pascal ist – nach Basic – wohl die beliebteste und verbreiteste höhere Programmiersprache. Mit ihr kann fast jede Programmieraufgabe gelöst werden. Deshalb haben wir Ihnen einen riesigen Pascal-Kurs anzubieten, der vor allem die Programmierpraxis erläutert. Zwei weitere Artikel gehen speziell auf Pascal mit dem C 64 und Turbo-Pascal auf dem C 128 ein. Eine Mischung aus Basic und Pascal wurde mit Comal geschaffen und wird daher von vielen Schulen, vor allem im norddeutschen Raum, gefördert und eingesetzt. Wir erklären Ihnen, wie mit Comal gearbeitet wird und warum diese Sprache so interessant ist. Weiterhin geben wir Ihnen ausführliche Informationen zu C (interessant vor allem für Systemprogrammierer), zu der maschinennahen Sprache Forth und zu Prolog, die vor allem im Zusammenhang mit künstlicher Intelligenz genannt wird.

Gerade für C 128-Besitzer tut sich mit dem Betriebssystem CP/M eine neue Welt auf. Sprachen wie Cobol und Fortran werden in der Industrie immer noch in großem Umfang eingesetzt. Wir zeigen Ihnen, was diese Oldtimer auf dem C 128 leisten und ob mit Ihnen vernünftig gearbeitet werden kann.

Eine Programmiersprache, die sich auf dem professionellen Markt durchgesetzt hat und zu den meistverkauften gehört, ist dBase. dBase ist datenbankorientiert und wird da eingesetzt, wo irgendwelche Daten zu verwalten sind. dBase ist relativ einfach zu erlernen, besitzt aber sehr leistungsfähige Befehle, die Sie mit anderen Programmiersprachen sehr umständlich oder gar nicht realisieren können. Wie man in und mit dBase programmiert, zeigen wir Ihnen in einem interessanten und ausführlichen Kurs.

Der zweite Teil dieses Sonderheftes befaßt sich mit der nach Basic am meisten genutzten Programmiersprache, mit Assembler. Und da haben wir einige besondere Leckerbissen für Sie herausgesucht. Sicher kennen Sie die in letzter Zeit immer häufiger auftauchenden Pull-down-Menüs. Schritt für Schritt erklären wir Ihnen, wie diese interessanten Menüs programmiert werden. Ein komplettes Programm zum Abtippen ist natürlich dabei. Über Sortierroutinen wurde schon viel geschrieben und es gibt eine ganze Reihe guter Programme. In diesem Sonderheft stellen wir Ihnen jedoch zwei Sortierprogramme vor, die in puncto Schnelligkeit und Vielseitigkeit alles bisher Dagewesene in den Schatten stellen. Lassen Sie sich überraschen! Wer schon etwas Gutes hat, hat nichts dagegen, etwas noch Besseres zu bekommen. Der Maschinensprachemonitor Promon ist ein Beispiel dafür. Er ist ein stark verbesserter und erweiterter SMON, den viele Basic- und Assemblerprogrammierer schätzen gelernt haben. Promon 64 wird auch Sie begeistern!

Selbstverständlich finden Sie neben diesen Grundlagen, Besprechungen und Super-Listings auch wieder eine ganze Menge Tips & Tricks, und wir hoffen, daß Ihnen dieses Sonderheft gefallen wird. Wir würden uns freuen, wenn Sie uns ein paar Zeilen schreiben würden.

Georg Klinge, leitender Redakteur

64'er SONDERHEFT

PROGRAMM-SERVICE

Bestellungen bitte an: Markt&Technik Verlag AG, Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar, Tel. 089/4613-0
Schweiz: Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Tel. 042/415656
Österreich: Bücherzentrum Meidling, Schönbrunner Straße 261, A-1120 Wien, Tel. 0222/833196, Microcomput-ique E. Schiller, Fasangasse 21, A-1030 Wien, Tel. 0222/785661, Ueberreuter Media Handels- und Verlagsgesellschaft mbH, Alser Straße 24, A-1091 Wien, Tel. 0222/481538-0
Bestellungen aus anderen Ländern bitte nur schriftlich an:
Markt&Technik Verlag AG, Abt. Buchvertrieb, Hans-Pinsel-Straße 2, 8013 Haar und gegen Bezahlung einer Rechnung im Voraus

TOP-Listings dieser Ausgabe:

Promon 64 – Super-Maschinensprachemonitor für den C64

Promon 64 ist ein vollkommen überarbeiteter SMON, der über einen riesigen und leistungsfähigen Befehlssatz verfügt. Neben den Befehlen des SMON können Sie HiRes-Grafiken suchen und ASCII-Tabellen eingeben. Illegale Opcodes können sowohl assembliert als auch disassembliert werden. Ein Diskettenmonitor, auf dem sich alle Befehle von Promon 64 anwenden lassen, ist integriert. Ein Spitzen-Werkzeug zu einem Super-Preis!

Alle Programme, die mit dem Diskettensymbol (■) im Inhaltsverzeichnis gekennzeichnet sind, gibt's auf Diskette.

1 Diskette für den C64
Bestell-Nr. L6 86 S12 D (sFr. 24,90/öS 299,-*) **DM 29,90***

Programme aus früheren 64'er-Ausgaben

Ausgabe	Bestell-Nr.		DM	sFr.	öS
10/86	L6 86 10D	Diskette	29,90*	24,90	299,00*
9/86	L6 86 09D	Diskette	29,90*	24,90	299,00*
8/86	L6 86 08D	Diskette	29,90*	24,90	299,00*
7/86	L6 86 07D	Diskette	29,90*	24,90	299,00*
6/86	L6 86 06D	Diskette	29,90*	24,90	299,00*
5/86	L6 86 05D	Diskette	29,90*	24,90	299,00*
4/86	L6 86 04D	Diskette	29,90*	24,90	299,00*
3/86	L6 86 03D	Diskette	29,90*	24,90	299,00*
2/86	L6 86 02D	Diskette	29,90*	24,90	299,00*
1/86	L6 86 01D	Diskette	29,90*	24,90	299,00*
12/85	L6 85 12D	Diskette	29,90*	24,90	299,00*
	L6 85 12K	Kassette	29,90*	24,90	299,00*
11/85	L6 85 11A	Diskette	29,90*	24,90	299,00*
10/85	L6 85 10A	Diskette	29,90*	24,90	299,00*
9/85	L6 85 09A	Diskette	29,90*	24,90	299,00*
8/85	L6 85 08A	Diskette	29,90*	24,90	299,00*
7/85	L6 85 07A	Diskette	29,90*	24,90	299,00*
6/85	L6 85 06A	Diskette	29,90*	24,90	299,00*
5/85	L6 85 05A	Diskette	29,90*	24,90	299,00*
4/85	L6 85 04A	Diskette	29,90*	24,90	299,00*
3/85	L6 85 03A	Diskette	29,90*	24,90	299,00*
2/85	L6 85 02A	Diskette	29,90*	24,90	299,00*
1/85	L6 85 01A	Diskette	29,90*	24,90	299,00*

Weitere Sonderhefte zum Thema Assembler und Programmiersprachen

Assembler für Anfänger und Fortgeschrittene
64'er-Sonderheft 8/85

Ein Standardwerk für alle, die sich für Maschinensprache interessieren. Dieses Heft vermittelt alle wichtigen und notwendigen Grundlagen für Einsteiger. Ein ausführlicher Kurs weist Sie speziell in die Programmierung des C64 ein, enthält viele und ausführlich erklärte Beispiele und beantwortet Fragen, die jeder hat oder später garantiert haben wird. Außerdem erhalten Sie in diesem Sonderheft alle Programme zum Abtippen, die Sie als Assembler-Programmierer brauchen werden und zwar durchweg von professioneller Qualität! Selbst wenn Sie schon einige Erfahrung haben, werden Sie die vielen Tips&Tricks-Listings begeistern.

1 Diskette für den C64
Bestell-Nr. L6 85 S8D (sFr. 24,90/öS 299,-*) **DM 29,90***

PEEKs&POKEs
64'er-Sonderheft 7/86

Jeder Assembler-Programmierer braucht Wissen über die Speicherstellen seines Computers. Einen Teil dieser Speicherstellen benötigt der C64 und natürlich noch mehr der C128 für eigene Aufgaben. Wenn man diese kennt, kann man sich eine Menge Arbeit und Ärger ersparen. In diesem Sonderheft werden deshalb alle wichtigen Adressen besprochen, erklärt und mit vielen praktischen Beispielen unterlegt. Es gibt nur ganz wenige Bücher, die, zugeschnitten auf den C64, erklären, wie man in Maschinensprache rechnet. Hier erfahren Sie, wie die mathematischen Routinen des C64 genutzt werden und was bei der Programmierung zu beachten ist. Auch dieses Sonderheft ist gespickt mit tollen Programmen zum Abtippen.

1 Diskette für den C64
Bestell-Nr. L6 86 S7D (sFr. 24,90/öS 299,-*) **DM 29,90***

* inkl. MwSt. Unverbindliche Preisempfehlung

Programme aus früheren 64'er-Sonderheften

Ausgabe	Bestell-Nr.		DM	sFr.	öS
10/86 C128	L6 86 S10 CD	Diskette	29,90*	24,90	299,00*
9/86 Floppy&Dateiverwaltung	L6 86 S9 CD	Diskette	29,90*	24,90	299,00*
8/86 Plus/4 und C16	L6 86 S8 CD	Diskette	29,90*	24,90	299,00*
	L6 86 S8 KC	4 Kassetten	34,90*	29,50	349,00*
	L6 86 S8 KV	Kassette	19,90*	17,00	199,00*
7/86 PEEKs & POKEs	L6 86 S7D	1 Diskette	29,90*	24,90	299,00*
6/86 Grafik	L6 86 S6D1	2 Disketten mit allen Programmen	34,90*	29,50	349,00*
	L6 86 S6D2	1 Diskette mit Giga-CAD-Demos	19,90*	17,00	199,00*
	L6 86 S6D3	3 Disketten mit allen Progr. und Demos	49,80*	43,50	498,00*
5/86 Grundwissen	L6 86 S5D	1 Diskette	29,90*	24,90	299,00*
4/86 Abenteuer	L6 86 S4D	2 Disketten	34,90*	29,50	349,00*
3/86 C16, C116, VC20, Plus/4	L6 86 S3 CD	1 Diskette für VC20 und C16/116	29,90*	24,90	299,00*
	L6 86 S3 KV	1 Kassette für VC20	19,90*	17,00	199,00*
	L6 86 S3 KC	1 Kassette für C16	19,90*	17,00	199,00*
2/86 Tips&Tricks	L6 86 S2D	Diskette	29,90*	24,90	299,00*
1/86 C128er	L6 86 S1D	Diskette	29,90*	24,90	299,00*
8/85 Assembler	L6 85 S8D	Diskette	29,90*	24,90	299,00*
	L6 85 S8K	Kassette	19,90*	17,00	199,00*
7/85 Professionelle Anwendungen	L6 85 S7D	2 Disketten	34,90*	29,50	349,00*
	L6 85 S7K	4 Kassetten	34,90*	29,50	349,00*
6/85 Top-Themen	L6 85 S6	2 Disketten	34,90*	29,50	349,00*
5/85 Floppy, Datasette	L6 85 S5D	Diskette	29,90*	24,90	299,00*
	L6 85 S5K	Kassette	19,90*	17,00	199,00*
4/85 Grafik	L6 85 S4A	Diskette	29,90*	24,90	299,00*
3/85 Spiele	L6 85 S3 A	2 Disketten	34,90*	29,50	349,00*
2/85 Abenteuerspiele	L6 85 S2	Diskette	34,90*	29,50	349,00*
1/85 Tips&Tricks (2. überarb. Auflage)	CB 023	Floppy-Utilities	29,90*	24,90	299,00*
	CB 024	Hilfsprogramme	29,90*	24,90	299,00*

Bitte verwenden Sie für Ihre Bestellung und Überweisung die eingeheftete Postgiro-Zahlkarte, oder senden Sie uns einen Verrechnungs-Scheck mit Ihrer Bestellung.
Sie erleichtern uns die Auftragsabwicklung, und dafür berechnen wir Ihnen keine Versandkosten.

INHALT

Vorwort

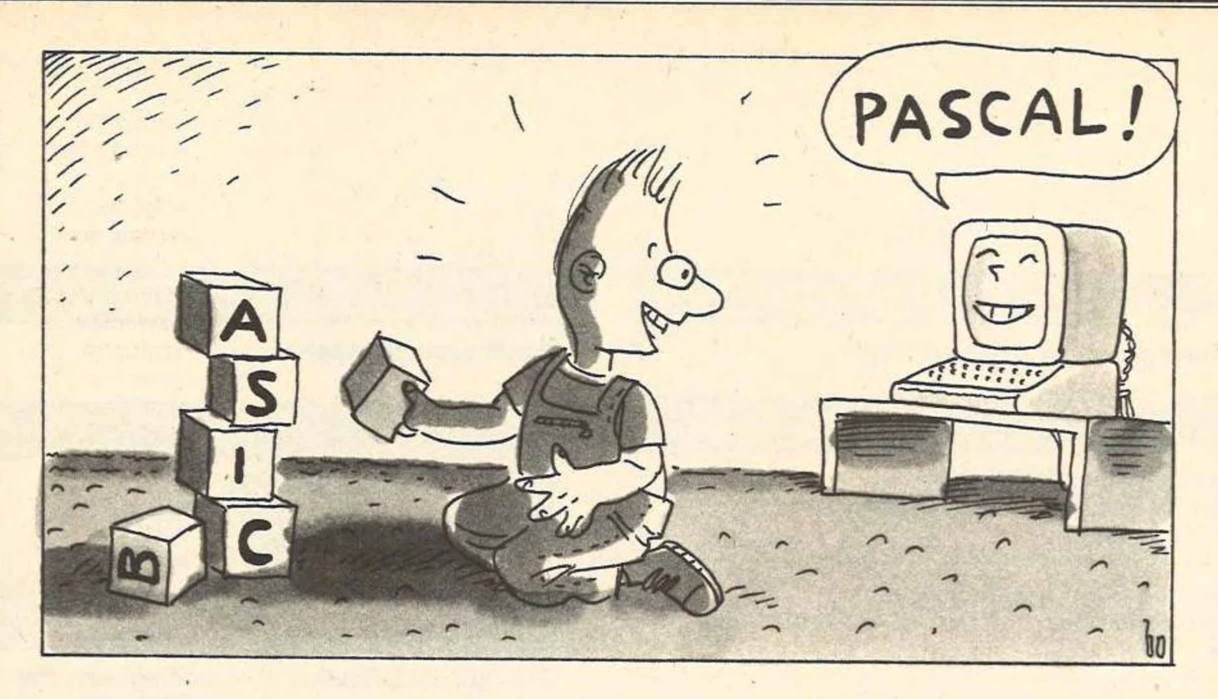
Programmiersprachen

CP/M

Bücher

Eingabehilfen

Tips & Tricks



Programmieren mit Struktur

Pascal ist eine Sprache, die auch bei den Heimcomputern zunehmend an Bedeutung gewinnt. Lernen Sie mit uns die Welt der strukturierten Programmierung kennen.

Wer im Mathematikunterricht gelegentlich aufgepaßt hat, dem wird der Name Pascal nicht unbekannt sein. Blaise Pascal war ein französischer Mathematiker, der von 1623 bis 1662 lebte und sich in der Wahrscheinlichkeitsrechnung, Kombinationslehre sowie mit dem Pascalschen Dreieck einen Namen machte. Nach ihm wurde auch eine höhere Programmiersprache benannt, die 1970 an der ETH Zürich von Nikolaus Wirth entwickelt wurde.

Diese Sprache bot im Gegensatz zu den anderen, damals sehr verbreiteten Sprachen Cobol und Fortran bessere Möglichkeiten zur Strukturierung von Anweisungen und Daten. Zudem war sie auf Grund ihres relativ geringen Sprachumfangs überschaubarer als ihre Konkurrenten. Speziell für die Anwendung auf Mikrocomputern erweiterte man das ursprünglich auf Großrechnern verwendete Standard-Pascal. Mit der neuen Version standen nun Erleichterungen zur grafischen Darstellung und Verarbeitung von Strings zur Verfügung. Man benannte es nach seinem Entwicklungsort, der Universität von Californien in San Diego, UCSD-Pascal.

Damit wurde ein Standard festgelegt, der heute von fast jedem Pascal-Compiler für Mikrocomputer akzeptiert wird. Dieser Kurs soll es ermöglichen, in die Welt von Pascal und somit in die strukturierte Programmierung einzusteigen. Viele Leser werden sich nun fragen, was es mit dem Begriff »strukturiertes Programmieren« auf sich hat. Das ist schnell erklärt. Strukturiertes Programmieren stellt eine Art von Programmierstil dar, der ein auch noch so komplexes Problem in einfache und klare Teilziele aufteilt. Die so programmierten Teillösungen werden dann zur Bewältigung des Gesamtproblems logisch zusammengefügt. Ein auf diese Art geschriebenes Programm ist für seinen Entwickler und besonders für andere besser les- und nachvollziehbar. Nachträgliche Änderungen, zum Beispiel Verbesserungen oder das Umschreiben auf andere Computersysteme, werden somit nicht mehr zu nervenzerreißenden Abenteuern.

Basic-Programmierer können bestimmt ein Lied davon singen, wenn sie sich in ihren Werken bereits nach einigen Tagen nicht mehr auskennen. Basic ist die wohl strukturloseste Sprache überhaupt und läßt Unsitten beim Programmieren zu, die bereits bei verhältnismäßig kleinen Programmen zu einem heillosen Durcheinander führen können. Aus diesem Grund ist Pascal eine gute Alternative für Basic-Programmierer, die ihre chaotische Art des Programmierens satt haben.

Aber auch Computerneulinge, die sich so etwas erst gar nicht angewöhnen wollen, sind mit Pascal gut bedient. So gibt es bei Pascal keine Zeilennummern und damit auch keine unübersichtlichen Sprünge quer durch das Programm. Ebenso müssen sämtliche Konstanten und Variablen, die irgendwann im Programm auch nur ein einziges Mal vorkommen sollen, vor Beginn des Hauptprogramms in einem besonderen Deklarationsteil festgelegt werden. Mit einfachem Drauflosprogrammieren am Bildschirm, wie bei Basic, dürfte man bei etwas größeren Programmen nicht mehr weit kommen. Es ist ein genaues Konzept ratsam, das die Einzelschritte der Lösung festlegt.

Sprache mit klarem Konzept

Pascal-Programme werden in der Regel auf dem Papier geschrieben und dann in den Computer eingegeben und getestet. Pascal ist außerdem eine Compilersprache, das heißt, das Programm muß nach der Eingabe zunächst vom Compiler in Maschinensprache übersetzt werden, bevor es ablaufen kann. Bevor wir Pascal-Programme schreiben können, müssen wir einige allgemeine Dinge lernen, die in Pascal von Bedeutung sind.

Der Zeichensatz, der der Sprache Pascal zur Verfügung steht, ist im allgemeinen das normale Alphabet und die Ziffern von null bis neun. Zusätzlich gibt es noch die Pascal-Sonderzeichen, wie zum Beispiel Rechenzeichen und Klammern. Aus ihnen setzen sich sämtliche Anweisungen zusammen. Bild 1 zeigt alle Zeichen auf einen Blick. Große und kleine Buchstaben haben in Pascal die gleiche Bedeutung und können nach Belieben benutzt werden. Eine Ausnahme

stellen die Druckerzeichen dar. Hier werden große und kleine Buchstaben unterschieden. In diesem Kurs werden zur Hervorhebung alle Pascal-Anweisungen und Standardnamen mit Großbuchstaben geschrieben. Verschiedene Namen dagegen sind klein geschrieben. Wenn Sie später selbst Pascal-Programme schreiben, ist es jedoch ohne Belang, ob Sie Groß- oder Kleinbuchstaben verwenden.

Weiterhin spielen Namen eine wichtige Rolle. Konstante, Variable und ganze Programmteile (Funktionen und Prozeduren) bekommen in Pascal einen Namen zur Identifizierung, mit dem sie dann ständig vom Programm aus aufgerufen werden können. Der Aufbau eines Namens ist einigen wichtigen Regeln unterworfen. Pascal-Namen sind beliebige Folgen von Buchstaben und Ziffern, die mit einem Buchstaben beginnen. Ein Beispiel wäre

```
meinersteswerk
```

Man muß dabei aber beachten, daß Pascal bei der Erkennung von Namen nur die ersten acht Zeichen berücksichtigt.

```
meinerstesprogramm
```

trägt für den Compiler demnach denselben Namen wie das erste Beispiel. Ebenso dürfen keine für Pascal bestimmten Sonderzeichen in Namen verwendet werden. Diese Sonderzeichen sind in Bild 1 aufgeführt und haben alle bestimmte Funktionen, die später noch erläutert werden. Das Leerzeichen dient bei Pascal der Trennung von Schlüsselwörtern und Namen und ist deshalb ebenfalls tabu, da der Compiler eine Verwendung mißverstehen könnte. Korrekte Pascal-Namen sind in Pascal zum Beispiel »egon, h2o2, k100« oder »x«. Weiterhin muß beachtet werden, daß kein Pascal-Schlüsselwort oder Standardname als eigener Name verwendet werden darf.

Die Befehle in Pascal werden auch als Schlüsselwörter bezeichnet, da sie keinem anderen Zweck dienen als dem ihnen zugewiesenen. Bild 2 zeigt eine Liste dieser Wörter, die man vor Namensgebung eigener Variable oder Konstanten aufmerksam studieren sollte, um späteren Ärger bei der Fehlersuche zu vermeiden. Ebenfalls reserviert sind alle Standardnamen, die in Bild 3 nach ihrer Zugehörigkeit geordnet sind. Sie haben schon programmierte Funktionen und können deshalb nicht mehr als eigene Namen fungieren.

Das Leerzeichen ist, wie erwähnt, als Zeichen für Namen nicht erlaubt, da es im Programm als Trennzeichen wirkt. Aufeinanderfolgende Schlüsselwörter und Namen müssen stets mit mindestens einem Leerzeichen getrennt werden, da es sonst zu Komplikationen bei der Übersetzung des Programms kommen kann. Das Einfügen von weiteren Leerzeichen ist jedoch erlaubt und in den meisten Fällen der Übersichtlichkeit des Programms dienlich. Durch eingefügte Leerzeichen kommen auch die für Pascal typischen Einrückungen von Befehlsblöcken zustande. Das Semikolon hat ebenfalls eine besondere Stellung. Jede abgeschlossene Anweisung muß mit einem Semikolon eindeutig beendet werden. Sollte man es einmal beim Programmieren vergessen, so kann das beim Übersetzen oder spätestens beim Programmablauf verheerende Folgen nach sich ziehen. Kann eine Anweisung durch ein darauf folgendes Schlüsselwort eindeutig als beendet angesehen werden, so muß das Semikolon nicht unbedingt stehen. Es ist jedoch zu empfehlen, sich anzugewöhnen, es immer zu schreiben.

Es ist möglich, zu Dokumentationszwecken an jeder Stelle eines Programms Kommentare einzufügen. Diese können beliebig lang sein und müssen in geschweiften Klammern stehen. Stehen dem Computer keine geschweiften Klammern zur Verfügung, ist ein Kommentar folgendermaßen zu schreiben:

```
(* dies ist ein kommentar *)
```

Wie Sie vermutlich bemerkt haben, ist das strikte Einhalten der pascalschen Regeln äußerst wichtig für das Entwickeln eines fehlerfreien Programms. Dies hat den Sinn, den Programmierer zu zwingen, bereits im voraus sein Programm komplett strukturiert durchzuplanen.

Geregelter Programmaufbau

Nun jedoch zum genauen Aufbau eines Pascal-Programms. Zu Anfang steht der Programmkopf, der zur Identifizierung des Programms dienen soll. Ihm folgt der Deklarationsteil, in dem sämtliche Konstante und Variable, die im Programm verwendet werden, vereinbart werden. Dann beginnt der Programmteil mit den einzelnen Unterprogrammen und dem Hauptprogramm. Der Programmkopf beginnt stets mit dem Schlüsselwort »PROGRAM«, dem der Programmname folgt. Nach Auswahl eines Pascal-gerechten Namens kann ein korrekter Programmkopf formuliert werden:

```
PROGRAM meinersteswerk;
```

Der Programmkopf ist eine vollständige Pascalanweisung und muß deshalb wie alle Anweisungen mit einem Semikolon abgeschlossen werden. Im Normalfall hat er keine Wirkung im Programm, sondern dient nur zur Benennung. Man kann ihn deshalb ohne Bedenken weglassen.

Obligatorisch dagegen ist der Deklarationsteil. In ihm werden Konstante und Variable, die im Programm verwendet werden sollen, vordefiniert und mit Namen versehen. Werden

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9
( ) + - * / = : ; . ^ '
```

Bild 1. Zeichen in Pascal

AND	END	NIL	SET
ARRAY	FILE	NOT	THEN
BEGIN	FOR	OF	TO
CASE	FUNCTION	OR	TYPE
CONST	GOTO	PACKED	UNTIL
DIV	IF	PROCEDURE	VAR
DO	IN	PROGRAM	WHILE
DOWNTO	LABEL	RECORD	WITH
ELSE	MOD	REPEAT	

Bild 2. Pascal-Schlüsselwörter

Konstanten:	Funktionen:	Prozeduren:
false	ABS	READ
true	ARCTAN	READLN
maxint	COS	WRITE
	EXP	WRITELN
Typen:	LN	PAGE
	SIN	RESET
integer	SQR	GET
real	SQRT	REWRITE
boolean	ROUND	PUT
char	TRUNC	NEW
text	ODD	DISPOSE
string	ORD	PACK
	CHR	UNPACK
Variable:	PRED SUCC	
input	EOLN	
output	EOF	

Bild 3. Standardnamen

keine Konstanten oder Variablen benötigt, so kann auch er selbstverständlich weggelassen werden.

Kommen wir nun zu den Konstanten und Variablen. Konstante sind Werte oder Zeichenfolgen, die sich während des Programmablaufs nicht verändern. Die Kreiszahl »Pi« ist beispielsweise eine solche Konstante. Konstante bekommen einen Namen, mit dem sie dann im Programm aufgerufen werden. Zu deren Definition ist das Schlüsselwort »CONST« vorgesehen. Eine Konstantendefinition könnte zum Beispiel so aussehen:

```
CONST pi = 3.14156;
```

Das Semikolon ist hier wieder das Schlußzeichen einer vollständigen Anweisung. Damit erhält die Zahl 3.14156 den Namen »pi«. Im Programm kann nun diese Zahl durch den Namen »pi« jederzeit aufgerufen und zu Berechnungen verwendet werden. »pi« kann aber jetzt keinen anderen Wert mehr zugewiesen bekommen. Es können auch bereits vorher definierte Konstante anderen Konstanten zugewiesen werden.

```
CONST pi = 3.14156;
      minuspi = -pi;
```

Konstante und Variable

Wie Sie sehen, muß das Wort CONST bei mehreren Konstantendefinitionen nur einmal am Anfang stehen. Jede einzelne Definition muß jedoch mit einem Semikolon abgeschlossen werden! Es gibt bei UCSD-Pascal eine Konstante, die schon vom Compiler vordefiniert ist. Sie hat den Namen »maxint« und steht für die größte darstellbare ganze Zahl. »maxint« ist einer der bereits erwähnten Standardnamen (Bild 3). Zwei weitere schon vom Compiler definierte Konstante sind logischer (boolean) Art. Sie haben die Namen »true« und »false« und entsprechen den logischen Werten »wahr« und »unwahr«. Die Verwendung von logischen Werten wird noch eingehend erläutert. Weiterhin kann eine Konstante auch aus einer Zeichenkette bestehen:

```
CONST name = 'Friedhelm';
```

Durch Aufruf der Konstanten »name« kann man nun die Zeichenkette »Friedhelm« an jeder Stelle im Programm verwenden. Wie aus dem obigen Beispiel ersichtlich ist, stehen in Pascal Zeichenketten oder allgemein Zeichen immer zwischen Apostrophen ('). Soll der Apostroph selbst als Druckzeichen verwendet werden, so muß man doppelte Apostrophe schreiben, wie zum Beispiel:

```
CONST apostroph = '''';
```

Anders als Konstante können Variable während des Programmablaufs verschiedene Werte annehmen. In Pascal lassen sich nicht alle Variable gleich verarbeiten. Durch Angabe des Typs legt man automatisch fest, was sie enthalten dürfen und was damit getan werden darf. Pascal kennt vier Standardtypen, die die Namen integer, real, boolean und char haben. UCSD-Pascal stellt zusätzlich noch einen weiteren Typen mit dem Namen string zur Verfügung. Es besteht außerdem noch die Möglichkeit, eigene Typen zu definieren. Eine Erklärung hierzu ist jedoch im Moment nicht sinnvoll und wird daher später nachgeholt. Zur Definition von Variablen benutzt man das Pascal-Schlüsselwort »VAR«, gefolgt vom Namen der Variablen plus der Typenbezeichnung, die sie bekommen soll (also integer, real, boolean, char oder string). Allgemein sieht eine Variablendefinition so aus:

```
VAR (Name): (Typ);
```

(Name) steht hier für einen beliebig wählbaren Namen. Mit (Typ) ist der entsprechende Variablentyp gemeint, den die Variable bekommen soll. Ein Deklarationsbeispiel wäre:

```
VAR ganzezahl: integer;
    zweitezahl: real;
```

Wie bei »CONST« ist bei mehreren Variablendeklarationen nur einmal das Wort »VAR« notwendig. Hat man mehrere Variable des gleichen Datentyps, so kann man beispielsweise schreiben:

```
VAR zahl1: integer;
    zahl2: integer;
```

Es ist aber auch erlaubt, Variable gleichen Typs durch Kommata getrennt hintereinander aufzulisten und anschließend den gemeinsamen Datentyp anzugeben:

```
VAR zahl1,zahl2: integer;
```

Die Liste der Variablennamen kann hierbei beliebig lang sein und über mehrere Bildschirmzeilen verlaufen. Die einzelnen Variablentypen sollen nun näher erläutert werden.

Integer-Werte sind alle positiven und negativen ganzen Zahlen, das heißt Zahlen ohne Nachkommastellen. Beispiele wären:

```
17, -24, 5349
```

Alle Zahlen, die keine Nachkommastellen haben, werden in Pascal als Integers interpretiert. Die größte, darstellbare ganze Zahl ist vom jeweiligen Pascal-Compiler abhängig. Sie ist in der Konstanten »maxint« bereits vordefiniert. Beim C 64 liegt der Bereich der Integer-Zahlen in der Regel zwischen -32767 und 32767. Die Deklaration von Integer-Variablen könnte zum Beispiel so aussehen:

```
VAR max,moritz: integer;
```

Die Variablen mit den Namen »max« und »moritz« werden damit als Integers definiert. »max« und »moritz« dürfen nurmehr ganze Zahlen beinhalten.

Der Datentyp real bezeichnet die Menge aller reellen Zahlen, das heißt aller gewöhnlichen positiven und negativen Zahlen wie zum Beispiel

```
5.8, -14.0, 3.14156
```

Alle Zahlen, die mindestens eine Nachkommastelle haben, werden in Pascal als Zahl des Typs real interpretiert. Eine Null zählt hierbei auch als Nachkommastelle, beispielsweise 14.0. Die Zahl 14 wird dagegen vom Compiler als Integer angesehen. Reelle Zahlen werden oft auch in der Exponentialschreibweise dargestellt. Mathematisch gesehen könnte man zum Beispiel die Zahl

```
27568.45
```

auch folgendermaßen schreiben:

```
2.756845 * 10^4
```

Diese Exponentialschreibweise, die man auch »wissenschaftliche Darstellung« nennt, hat insbesondere bei sehr großen und sehr kleinen Zahlen gewisse Vorteile. Pascal beherrscht auch diese Art der Darstellung. Für den Ausdruck »* 10^« wird lediglich der Buchstabe »e« (für Exponent) verwendet. So würde unser letztes Beispiel in Pascal-Exponentschreibweise so lauten:

```
2.756845e+4
```

Diese Art von Darstellung wird vom Computer anstandslos akzeptiert. Hat man eine Variable als real deklariert, so kann sie alle reellen Werte annehmen. Die korrekte Definition einer Variablen des Typs real wäre beispielsweise

```
VAR länge: real;
```

Variable des Typs boolean können nur zwei Werte annehmen: »true« und »false«, die beiden Wahrheitswerte. Bei beiden handelt es sich um bereits vordefinierte Konstante. Ein Definitionsbeispiel hierzu wäre:

```
VAR bestanden: boolean;
```

Die Variable »bestanden« kann jetzt nur die Werte »true« oder »false« annehmen. Was man mit solchen Variablen anfangen kann, wird zu einem späteren Zeitpunkt eingehend erklärt.

Der Datentyp char umfaßt alle dem Pascal-Compiler zur Verfügung stehenden Druckzeichen (siehe Bild 1). Eine so deklarierte Variable kann nur eines der Druckzeichen aufnehmen; in diesem Fall können auch das Leerzeichen und die Pascal-Sonderzeichen Druckzeichen sein. Analog zu den anderen Datentypen wird eine Variable des Typs char zum Beispiel so definiert:

```
VAR buchstabe: char;
```

Benötigt man Variable, die mehr als nur ein Zeichen, also auch Zeichenketten, enthalten können, so bietet UCSD-Pascal den zusätzlichen Datentyp string an. Ein Beispiel hierfür wäre:

```
VAR wohnort: string;
```

Die Variable »wohnort« hat in diesem Fall eine variable Länge, das heißt, es können Zeichenketten mit verschiedener Länge darin abgelegt werden. Es ist aber auch beim Datentyp string möglich, eine feste Länge anzugeben. Dies geschieht durch eine Längenangabe in eckigen Klammern direkt hinter dem Wort »string«, wie zum Beispiel

```
VAR wohnort: string[20];
```

Die maximale Zeichenlänge der Variablen »wohnort« beträgt jetzt nur 20 Zeichen. Mit diesen Informationen können Sie bereits jetzt einen perfekten Programmkopf und Deklarationsteil in Pascal schreiben, zum Beispiel so:

```
PROGRAM meinersteswerk;

 CONST ziffern = '0123456789';
       sternchen = '*';
       plus = 456;

       minus = -plus;

 VAR   buchstabe: char;
       betrag,summe: real;
       zaehler: integer;
       nachname: string;
       antwort: boolean;
```

Doch der Deklarationsteil allein macht noch kein Pascal-Programm. Kommen wir deshalb zum ausführenden Teil eines Programms, dem Anweisungsteil. Er beginnt stets mit dem Schlüsselwort »BEGIN« und muß mit dem Wort »END« und einem darauf folgenden Punkt abgeschlossen werden.

Operatoren:	
+	Addition
-	Subtraktion
*	Multiplikation
DIV	ganzzahlige Division
MOD	Rest einer ganzzahligen Division
Funktionen:	
SQR(x)	Quadrat von x
ABS(x)	Absolutwert von x
PRED(x)	Vorgänger von x
SUCC(x)	Nachfolger von x

Bild 4. Integer-Operatoren

Operatoren:	
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
Funktionen:	
SQR(x)	Quadrat von x
ABS(x)	Absolutwert von x
SQRT(X)	Quadratwurzel von x
EXP(x)	Exponentation von x zur Basis e
LN(x)	natürlicher Logarithmus von x
SIN(x)	Sinus von x
COS(x)	Cosinus von x
ARCTAN(x)	Arcusstangens von x

Bild 5. Real-Operatoren

Dazwischen stehen nun die einzelnen Anweisungen, die das Programm ausführen soll. »BEGIN« und »END« können auch ohne dazwischen stehende Befehle unmittelbar hintereinander stehen. Damit kann man auch schon sein allererstes Programm schreiben, das wie folgt lautet:

```
PROGRAM adam;

 BEGIN END.
```

Dieses Programm bewirkt selbstverständlich nichts, da keine Anweisungen zwischen »BEGIN« und »END« stehen. Programmtechnisch ist dies aber bereits ein perfektes Pascal-Programm.

Damit kommen wir zu den Befehlen und Operatoren von UCSD-Pascal. Entsprechend ihres Datentyps bietet Pascal zur Verarbeitung von Variablen und Werten eine Reihe von Operatoren und Funktionen an, die hauptsächlich arithmetischer Art sind. Bild 4 zeigt sämtliche Operatoren und Funktionen mit ihren Bedeutungen, die auf Werte und Variable des Typs integer angewandt werden können.

Neben den arithmetischen Operatoren für die Addition, Subtraktion und Multiplikation gibt es die Operatoren »DIV« und »MOD«. »DIV« dividiert zwei ganze Zahlen ohne Berücksichtigung des Restes und liefert nur das ganzzahlige Ergebnis. Als Umkehrung dazu bestimmt »MOD« den Rest einer solchen Division. Wie diese Operatoren arbeiten, soll in einigen Beispielen gezeigt werden:

```
18 DIV 4   ergibt 4
11 DIV 2   ergibt 2
 7 DIV 9   ergibt 0

18 MOD 4   ergibt 2
11 MOD 2   ergibt 1
 7 MOD 9   ergibt 7
```

Wie Sie in Bild 4 sehen können, dürfen auch vier Standardfunktionen, die wiederum Integer-Werte liefern, angewandt werden. Was Funktionen genau sind, werden wir später noch erfahren. Merken wir uns vorerst, daß sie durch Angaben von Argumenten, das heißt bestimmten Werten, andere Werte entsprechend ihrer Funktion liefern. sqr() für das Quadrat und abs() für den Absolutbetrag eines Integerwertes haben die gleichen Funktionen wie in der Mathematik. Auch hier sollen einige Beispiele zur Verdeutlichung dienen:

```
sqr(5)     ergibt 25
sqr(-8)    ergibt 64

abs(16)    ergibt 16
abs(-19)   ergibt 19
```

»pred()« und »succ()« hingegen bestimmen den Vorgänger und Nachfolger einer Integer-Zahl. Welcher Wert Vorgänger oder Nachfolger ist, wird durch die vom Datentyp festgelegte Ordnung bestimmt. Im Falle der Integer-Zahlen werden Vorgänger und Nachfolger der Größe nach ermittelt. Beispiele seien hier:

```
pred(9)    ergibt 8
pred(5)    ergibt 4
pred(-8)   ergibt -9

succ(9)    ergibt 10
succ(5)    ergibt  6
succ(-8)   ergibt -7
```

Für die Zahlen und Variablen des Typs real, also für alle reellen Zahlen, stehen etwas mehr Funktionen zur Verfügung. Bild 5 zeigt sie in einer Tabelle zusammengefaßt. Im Gegensatz zu den Integer-Operatoren kann bei den Zahlen des Typs real die normale Division mit Nachkommastellen angewandt werden. Eine Verwendung von »DIV« und »MOD« ist dagegen unzulässig. Beispiele sind:

```
5.2 - 1.7 ergibt 3.5
18.0 / 4.0 ergibt 4.5   (dagegen 18 DIV 4 ergibt 4)
```

Die Palette der Funktionen hat sich erweitert. Außer der bekannten Quadrat- und Absolutfunktion stehen hier auch die Quadratwurzel-, die »e«-Funktion, der natürliche Logarithmus und die trigonometrischen Funktionen Sinus, Cosinus und Arcustangens zur Verfügung. Die Funktionen »pred()« und »succ()« liefern beim Datentyp real keine eindeutigen Ergebnisse und sind deshalb bei diesem Typ nicht gestattet. Statt dessen stehen zwei neue Funktionen zur Verfügung, die eine reelle Zahl in einen Integer-Wert verwandeln. Die Funktion »trunc()« schneidet alle Nachkommastellen einer reellen Zahl ab und hinterläßt nur den ganzzahligen Teil. »round()« hingegen rundet eine reelle Zahl auf die nächste ganze Zahl. Der Unterschied der beiden Funktionen wird in einigen Beispielen deutlich:

```
trunc(3.1415)    ergibt 3
trunc(5.7594)    ergibt 5

round(5.7594)    ergibt dagegen 6, da bei
                 den Nachkommastellen ab 0.5
                 aufgerundet wird.
round(2.1718)    ergibt 2
```

In Bild 6 sehen Sie die Operatoren und Funktionen, die für Werte und Variable des Typs boolean gedacht sind.

Dies sind eigentlich nur drei logische Operatoren mit den Bezeichnungen »NOT« (Negation), »AND« (Konjunktion) und »OR« (Disjunktion). Aber auch hier dürfen einige Beispiele nicht fehlen:

```
      NOT true    ergibt false
 true AND false   ergibt false
 true OR  true    ergibt true
```

Beim Datentyp boolean sind die beiden Funktionen »pred()« und »succ()« wieder gestattet, jedoch nur in den beiden folgenden Fällen:

```
pred(true)    ergibt false
succ(false)   ergibt true
```

»pred(false)« und »succ(true)« sind dagegen nicht definiert und geben daher auch kein Ergebnis. Eine weitere Funktion, die als Argument zwar einen Wert des Typs Integer benötigt, als Ergebnis aber einen boolschen Wert liefert, ist »odd()«. Sie ist »true«, wenn der Wert in den Klammern eine ungerade Zahl ist. Für eine gerade Zahl ergibt die Funktion »false«. Beispiele hierfür sind:

```
odd(15)    ergibt true
odd(-44)   ergibt false
```

Für den Datentyp char gibt es keine der oben genannten Operatoren zur Verarbeitung. Es können jedoch die sogenannten relationalen Operatoren angewandt werden. Diese werden später noch ausführlich besprochen. Kommen wir zunächst zu den für den UCSD-Standard üblichen Datentyp string. Auch dafür können keine der bereits erwähnten Operatoren und Funktionen verwendet werden. UCSD-Pascal bietet vielmehr besondere Funktionen und Prozeduren, die nur für Strings gedacht sind. In Bild 7 sind sie tabellarisch aufgeführt.

Was man damit anfangen kann, ist speziell zur Stringverarbeitung vorgesehen. Zum Verständnis der Grundkenntnisse von Pascal sind diese Funktionen jedoch nicht nötig und werden deshalb an dieser Stelle übergangen. Sie werden später in einem eigenen Teil besprochen, der sich ausschließlich mit den UCSD-Standardfunktionen beschäftigt. Mit Hilfe der Tabellen in Bild 4 bis 6 können Sie jetzt bereits einfache Berechnungen in Pascal anstellen.

Im Anweisungsteil können den vordefinierten Variablen Werte zugewiesen werden. Dies geschieht mit dem Zuweisungsoperator »:=« , wie zum Beispiel:

```
zahl := 46;
zeichen := 'a';
```

Die Variablen dürfen aber nur Werte gemäß ihres vorbestimmten Datentyps erhalten. So lassen sich einer Integer-Variable nur Integer-Werte zuweisen, einer Variablen des Typs real dagegen nur reelle Zahlen. Wenn man den Datentyp bei der Zuweisung nicht beachtet, kann einem das der Pascal-Compiler recht übelnehmen. Um diese wichtige Regel zu veranschaulichen, sei ein kleines Beispiel aufgeführt. Folgende Variablen werden definiert.

```
VAR ganzezahl: integer;
    reellezahl: real;
    zeichen: char;
    kette: string;
```

Jetzt können diesen Variablen entsprechende Werte zugewiesen werden, zum Beispiel so:

```
BEGIN
    ganzezahl := 248;
    reellezahl := 3.14156;
    zeichen := 'a';
END.
```

Unkorrekt dagegen wäre folgendes:

```
BEGIN
    ganzezahl := 2.7182818;
    reellezahl := '1';
    zeichen := 40.1;
END.
```


Es können aber auch ganze Ausdrücke zugewiesen werden. Diese Ausdrücke dürfen sich aus Operanden, Operatoren, Klammern und Funktionen zusammensetzen. Ausdrücke können zum Beispiel so aussehen:

```
15.2 + 9.98
(maxint - 30000) * (45 DIV 3)
false OR (true AND NOT true)
2-sqr(28 * 4)
```

Die Berechnung der Ausdrücke erfolgt durch besondere Regeln. Eine recht einfache ist die aus der Mathematik bekannte Regel »Punkt vor Strich«. Sie besagt, daß die Multiplikation und Division vor der Addition und Subtraktion ausgeführt wird. Die Zeichen »*« oder »/« haben also eine höhere Priorität als »+« und »-«. Man sagt auch, daß »*« und »/« stärker »binden«. Die Regeln in Pascal sind hauptsächlich die arithmetischen Bindungsregeln. So haben Klammern und die logische Negation »NOT« die höchste Priorität und werden zuerst berechnet. Danach kommen die Multiplikationsoperatoren »*,/,DIV,MOD,AND«, gefolgt von den Additionsoperatoren »+,-,OR« und letztendlich die Vergleichsoperatoren (Bild 8), deren Bedeutung noch erklärt wird. Operatoren der gleichen Priorität werden in der Reihe ihres Auftretens verarbeitet. Es muß darauf geachtet werden, daß die verwendeten

Operatoren:	
NOT x	Negation
AND	Konjunktion
OR	Disjunktion
Funktionen:	
ODD(x)	Test ob x gerade
PRED(x)	Vorgänger von x
SUCC(x)	Nachfolger von x

Bild 6. Boolean-Operatoren

Funktionen:	
CONCAT(string1,string2)	Verbinden von string1 und 2
COPY(string,anfang,länge)	Kopieren eines Teilstrings
POS(Musterstring,Quellstring)	Test auf identische Strings
Prozeduren:	
INSERT(string,Zielstring,anfang)	Einfügen eines Strings
DELETE(Zielstring,anfang,anzahl)	Löschen eines Teilstrings

Bild 7. Stringfunktionen

Operatoren auch tatsächlich nur mit den Werten des richtigen Datentyps verknüpft werden. So wäre ein Ausdruck wie

```
1.987 DIV 0.27
```

nicht erlaubt, da »DIV« nur mit Integer-Werten verknüpft werden darf. Eine Ausnahme bilden die Operatoren und Funktionen für Zahlen des Typs real. Sie dürfen auch in Verbindung mit Integer-Zahlen angewandt werden. Ein Integer-Wert wird dann vom Computer selbständig in einen Real-Wert umgewandelt. Umgekehrt geht dies aber nicht, wie das obige Beispiel mit »DIV« zeigt. Man muß ebenfalls darauf achten, daß der Gesamtausdruck rechts vom Zuweisungsoperator »:=« im Ergebnis den gleichen Datentyp hat, wie die Variable links davon. Wir können zum Beispiel folgende Variablen deklarieren:

```
VAR a,b,c,d: integer;
    e,f: real;
    g,h,i: boolean;
```

So wären die Variablenzuweisungen

```
e := a - b / d
g := g AND NOT (h OR i)
f := c MOD b
```

durchaus in Ordnung. Anders bei diesen Zuweisungen:

```
a := e - f
g := false OR (NOT i + d)
d := b / c * a
```

Die dürfte der Compiler wohl nicht akzeptieren. Man sollte sich besonders gut mit den Datentypen und deren zulässigen Operatoren vertraut machen, da die Anwendung von nicht zulässigen Operatoren eine häufige Fehlerquelle beim Programmieren ist.

```
PROGRAM berechnung;
CONST pi = 3.14156;
VAR radius,umfang: real;
  BEGIN
    radius := 2.5;
    umfang := 2 * pi * radius;
  END.
```

Listing 1. Einfache Berechnungen

Nach Studium des bisherigen Teils unseres Pascal-Kurses sind Sie nun imstande, ein kleines Pascal-Programm zu schreiben, in dem auch wirklich etwas passiert (Listing 1). Damit haben wir zwar ein Programm, das Aktionen ausführt, doch fehlt noch die Verbindung nach außen. Es gab bisher noch keine Möglichkeiten, die bearbeiteten Daten auf dem Bildschirm sichtbar zu machen oder dem Programm Daten und Werte über die Tastatur mitzuteilen. Dazu hat Pascal spezielle Anweisungen parat, die die Ein- und Ausgabe von Daten ermöglichen. Es handelt sich um die sogenannten Standardprozeduren »READ« und »WRITE«. Die »READ«-Anweisung erlaubt dem Anwender die Eingabe von Daten während des Programmablaufs. Dieser Befehl ist dem INPUT-Statement von Basic sehr ähnlich. Mit

```
READ(<Variablenname>);
```

lassen sich Variablen von der Tastatur aus Werte zuweisen, die dann vom Programm weiterverarbeitet werden. Gelangt das Programm an eine »READ«-Anweisung, so wird es in der Regel unterbrochen und der Computer wartet auf eine Eingabe von der Tastatur, die mit der Taste <RETURN> abgeschlossen werden muß. Unter der Voraussetzung, daß der verwendete Variablenname vorher ordnungsgemäß deklariert wurde, können Eingaben mit »READ« beispielsweise folgendermaßen geschehen:

```
READ(zahl);
READ(zeichen);
READ(text);
```

Die angegebenen Namen müssen Variable vom Typ integer, real, char oder (speziell für UCSD-Pascal) string sein. Die Eingabe muß selbstverständlich zum Typ der Variablen passen, sonst kommt es zu einer Fehlermeldung. So ist es beispielsweise unmöglich, in eine als integer deklarierte Variable Zeichen einzugeben. Um verschiedene Daten auf dem Bildschirm sichtbar zu machen, ist die »WRITE«-Prozedur zu verwenden. Sie hat eine ähnliche Syntax wie »READ«:

```
WRITE(<Ausdruck>);
```

Ein Ausdruck kann hierbei aus Variablen, Zeichen, Zeichenketten oder ganzen Rechenausdrücken bestehen. Einige Beispiele machen dies deutlich:

```
Befehl                  Ausgabe
WRITE(3.14156);         3.14156
WRITE('Ausgabe');       Ausgabe
WRITE(ergebnis);        (Wert der Variablen
                        "ergebnis")
WRITE(7 + 12 * 2);      31
WRITE(max * moritz);    (Wert aus Multiplikation der
                        Variablen "max" und "moritz")
```

Es können Variable und Werte aller Datentypen ausgegeben werden. Selbst Werte des Typs boolean können mit »WRITE« auf den Bildschirm geschrieben werden. Es erscheint dann jeweils die Ausgabe »true« oder »false«. Die Ausführung der »WRITE«-Anweisung geschieht immer an der momentanen Cursorposition. Zwei Zeichenketten würden also unmittelbar hintereinander auf dem Bildschirm erscheinen. Die Anweisungen

```
WRITE('Erste Zeichen');
WRITE('Zweite Zeichen');
```

würden demnach zu folgendem Ausdruck führen:

```
Erste ZeichenZweite Zeichen
```

Wünscht man, daß die Ausgabe in einer neuen Zeile beginnen soll, muß man die Anweisung »WRITELN« benutzen. Man nennt sie auch »Write-line«-Anweisung, die bewirkt, daß ein »EOLN« (End-of-Line-Zeichen) an den Computer gesendet wird. Das heißt für ihn, weitere Ausgaben in einer neuen Zeile fortzusetzen.

```
WRITELN;
```

bewirkt einen Zeilenvorschub zum Anfang der nächsten Zeile. Man kann aber »WRITELN« auch direkt zur Ausgabe verwenden, wie zum Beispiel:

```
WRITELN('Dies ist ein Satz');
```

In diesem Fall wird der Ausdruck innerhalb der Klammern ausgegeben. Anschließend wird das »EOLN« gesendet, welches, wie bereits erwähnt, einen Zeilenvorschub bewirkt. Das gleiche Ergebnis hätten folgende Anweisungen:

```
WRITE('Dies ist ein Satz'); WRITELN;
```

Analog dazu gibt es auch den »READLN«-oder »Read-Line«-Befehl. Dieser Befehl bewirkt, daß mit der Eingabe in der nächsten Bildschirmzeile fortgefahren wird. Man schreibt einfach:

```
READLN;
```

Wie bei der »WRITELN«-Anweisung können auch hier Eingabe-Variablen dabeistehen. Beispielsweise läßt

```
READLN(zahl);
```

eine Eingabe der Variablen mit dem Namen »zahl« zu und liest danach das »EOLN«. So könnte man dafür ebenfalls schreiben:

```
READ(zahl); READLN;
```

Neben der einfachen Ausgabe von Werten und Zeichen hat der »WRITE«-Befehl noch eine weitere sehr nützliche Eigenschaft. Durch Angabe weiterer Faktoren in der »WRITE«-Anweisung kann man seine Werte in einem bestimmten Format ausgeben lassen. Die Anweisung

```
WRITE(a:f);
```

bewirkt, daß der Ausdruck »a« durch genau »f«-Zeichen dargestellt wird. Benötigt »a« eine größere Zeichenanzahl als in »f« angegeben, so wird »a« mit entsprechend mehr Zeichen ausgedruckt. Es werden also keine Zeichen abgeschnitten oder nicht ausgegeben. Ist der Ausdruck jedoch kleiner als

»f«-Stellen, werden die fehlenden Zeichen durch vorangestellte Leerzeichen ausgefüllt. Wie das in der Praxis aussieht, soll mit einigen Beispielen verdeutlicht werden. Das Zeichen »L« entspricht hierbei einem ausgegebenen Leerzeichen.

```
Anweisung             Ausgabe
WRITE(24680:6);       L24680
WRITE(-13579:9);      LLL-13579
WRITE(10000:3);       10000
WRITE('Text':6);      LLText
WRITE('Text':2);      Text
```

Sollte der auszugebende Wert vom Datentyp real sein, müssen einige Besonderheiten beachtet werden. Ist »f« angegeben, so erfolgt die Ausgabe der Zahl in Exponentialschreibweise mit mindestens einem vorangestellten Leerzeichen. Wie diese Schreibweise aussieht, wurde bei der Erklärung des Datentyps real bereits angeschnitten. Hierzu wieder einige Beispiele:

```
Anweisung             Ausgabe
WRITE(56.78:14);      L5.67800e+0001
WRITE(-56.78:15);     L-5.67800e+0001
WRITE(-56.78:11);     L-5.6e+0001
```

Durch Angabe eines weiteren Parameters kann man bei Daten des Typs real zusätzlich die Genauigkeit der Ausgabe festlegen. Die Anweisung dazu lautet:

```
WRITE(a:f:g);
```

Mit dem zusätzlichen Parameter »g« wird die Stellengenauigkeit bestimmt. Der Ausdruck wird somit mit »f«-Stellen Länge und einer Genauigkeit von »g«-Stellen hinter dem Dezimalpunkt angegeben. Auch hier wieder einige Beispiele:

```
Anweisung               Ausgabe
WRITE(-56.78:9:3);      LL-56.780
WRITE(-56.78:4:4);      L-56.7800
WRITE(-5.678e+1:9:1);   LLLL-56.7
```

Ausgabe mit Komfort

Damit sind die Möglichkeiten der »READ«- und »WRITE«-Anweisungen noch nicht erschöpft. Über die komfortable formatierte Ausgabe hinaus läßt es Pascal auch zu, mehrere Ausdrücke mit nur einem »WRITE«-Befehl auszugeben. Die einzelnen Ausdrücke müssen nur durch Kommata voneinander getrennt werden. Diese Ausdrücke können den verschiedensten Datentypen angehören. So ist zum Beispiel folgender Befehl vollkommen korrekt:

```
WRITE('Die Summe beträgt ',100,' DM');
```

Die Ausgabe würde folgendermaßen lauten:

```
Die Summe beträgt 100 DM
```

Die Wirkung ist hierbei die gleiche, wie beim Ausführen einzelner »WRITE«-Befehle hintereinander. Es sind ebenfalls verschiedene Formate gestattet, wie zum Beispiel:

```
WRITE(12.56:8:3 , 98765:6);
```

Auch die »READ«-Anweisung ist ähnlich komfortabel. Es ist auch hier möglich, mehrere Eingabevariablen in nur einem »READ«-Befehl unterzubringen. Auch sie müssen durch Kommatas getrennt werden und können unterschiedlichen Typs sein. Wichtig ist nur, daß die jeweiligen Tastatureingaben mit den jeweiligen Variablentypen übereinstimmen.

```
READ(wert1,wert2,wert3,name);
```

ist zum Beispiel durchaus möglich. Wie bei »WRITE« bewirkt dies das gleiche, wie einzeln hintereinander geschriebene »READ«-Anweisungen. Dieselben Regeln gelten auch für die Befehle »WRITELN« und »READLN«. Die Anweisung

```
WRITELN(wert1,wert2);
```

ist äquivalent zu

```
WRITE(wert1,wert2); WRITELN;
```

Ebenso ist

```
READLN(wert1,wert2);
```

gleichbedeutend mit

```
READ(wert1,wert2); READLN;
```

```
PROGRAM wurzel;
VAR zahl1,zahl2, wurzel: real;
  BEGIN
    READLN(zahl1);
    READLN(zahl2);
    WRITELN('Die Summe ist ',zahl1 + zahl2);
    wurzel := SQRT(zahl1 + zahl2);
    WRITELN('Die Wurzel daraus ist ',wurzel:6:4);
  END.
```

Listing 2. Berechnungen mit Ein- und Ausgabe

Nachdem Sie diese Unmengen an Information verdaut haben, können Sie jetzt bereits Ihre ersten Programme schreiben, die schon die Ein- und Ausgabe von Werten und Zeichen beinhalten. Außerdem wissen Sie, wie man in Pascal rechnet und sind damit schon in der Lage, kleine mathematische Probleme in Pascal zu formulieren, wie zum Beispiel das Programm in Listing 2, welches zwei reelle Zahlen addiert und daraus die Wurzel zieht. Die Ausgabe erfolgt formatiert auf vier Stellen hinter dem Dezimalpunkt genau.

Bisher können Sie Pascal-Programme schreiben, die Daten von der Tastatur aufnehmen, mit ihnen Berechnungen durchführen und schließlich die Ergebnisse oder anderes auf dem Bildschirm ausdrucken. Dabei werden alle Befehle nach dem Schlüsselwort »BEGIN« der Reihe nach ausgeführt, bis das Wort »END.« erreicht ist. Für einfache Rechnungen ist dies auch vollkommen ausreichend. Komplexere Probleme bedürfen jedoch verschiedener Steuermöglichkeiten, die bestimmen, wann welche Anweisungen unter welchen Bedingungen abzulaufen haben. Hinzu kommen Wiederholungen von Programmabschnitten, die öfter als einmal gebraucht werden. Pascal bietet eine Reihe von Anweisungen, die dies übernehmen.

Viele Programmier-Probleme machen es nötig, daß eine oder mehrere Anweisungen mehrmals hintereinander ausgeführt werden müssen, wie zum Beispiel die Eingabe von mehreren gleichartigen Daten. Mit den Möglichkeiten, die wir bereits gelernt haben, hieße das, daß wir diese Anweisungen so oft hintereinander im Programm aufschreiben müßten, wie es der Zahl der gewünschten Wiederholungen entspricht. Eine mühselige und vor allem Speicherplatz raubende Methode. Um dies zu vermeiden, gibt es in Pascal die sogenannte »Zählschleife«. Sie erlaubt ein beliebig langes Wiederholen von Anweisungen. Sie wird auch FOR-Schleife genannt und hat in Pascal folgende Syntax:

```
FOR (Kontrollvariable) := (a) TO (e)
    DO (Anweisung);
```

Als Kontrollvariable kann eine beliebige Variable verwendet werden, bei deren Datentyp der Nachfolgerwert eindeutig bestimmt werden kann. Das heißt, daß die Funktion »succ()« definiert sein muß. Dies wären, wie wir an den einzelnen Operatorentabellen in Bild 4-6 erkennen können, die Typen integer, char und boolean. Auf keinen Fall kann man die Datentypen real und string benutzen. »a« und »e« sind die Anfangs- und Endwerte der Schleife. Sie dürfen aus Werten, Variablen oder ganzen Ausdrücken bestehen. Man muß aber darauf achten, daß sie dem Datentyp der Kontrollvariable entsprechen. (Anweisung) stellt in diesem Fall schließlich die Anweisung dar, die wiederholt werden soll. In der Regel kann jede beliebige Anweisung hinter »DO« stehen. Selbst die Kontrollvariable darf dabei zu Berechnungen benutzt werden. Eine Veränderung derselben innerhalb der Anweisung ist in Pascal jedoch verboten. Eine verbotene FOR-Schleife ist zum Beispiel:

```
FOR faktor := 1 TO 20 DO faktor := sqr(faktor);
```

Sie verändert die Kontrollvariable »faktor« während des Schleifenablaufs. Mit einer kleinen Änderung wird diese

Schleife wieder syntaktisch korrekt:

```
FOR faktor := 1 TO 20 DO loesung := sqr(faktor);
```

Statt der Kontrollvariablen »faktor« wird nun das Ergebnis der Variablen »loesung« zugewiesen. Da »faktor« auch als Kontrollvariable als Wert verwendet werden darf, kann das zweite »faktor« stehengelassen werden. Trifft der Computer auf einen »FOR«-Befehl, nimmt die Kontrollvariable den Anfangswert »a« an. Danach wird die Anweisung nach »DO« ausgeführt. Der Computer erhöht nun den Wert der Kontrollvariablen auf ihren Nachfolgewert und führt ein weiteres Mal die Anweisung hinter »DO« aus. Die letzten beiden Schritte werden so lange wiederholt, bis der Endwert »e« erreicht ist. Anschließend wird die Anweisung ein letztes Mal aufgerufen, die Schleife beendet und der nächste Befehl dahinter abgearbeitet. Sollte der Anfangswert zu Beginn der Schleife schon größer als der Endwert sein, wird die Schleife gar nicht ausgeführt. Möchte man mehrere Anweisungen, also einen ganzen Anweisungsblock, wiederholen, so ist dies mit »FOR...DO« auch möglich. Dabei werden die zu wiederholenden Befehle hinter »DO« mit dem Schlüsselwort »BEGIN« begonnen und mit »END« abgeschlossen. Die FOR-Schleife sieht nun so aus:

```
FOR (Kontrollvariable) := (a) TO (e) DO BEGIN
    (Anweisungen) END;
```

Nun werden Sie sagen, daß die beiden Worte »BEGIN« und »END« doch bereits für den Beginn und das Ende des Gesamtprogramms reserviert sind. Dies ist aber nicht ganz richtig. Pascal verwendet »BEGIN« und »END« allgemein zur Kennzeichnung eines zusammengehörenden Anweisungsblocks, der auch Verbundanweisung genannt wird. Somit wird auch das Hauptprogramm, also der gesamte Anweisungsteil eines Pascal-Programms, als eine einzige Verbundanweisung verstanden. Es können hierbei beliebig viele Anweisungen zwischen »BEGIN« und »END« stehen. In Listing 3 sehen Sie ein Beispiel für die Verwendung einer FOR-Schleife.

Pascal erlaubt auch verschachtelte Schleifen, das heißt es darf auch innerhalb einer »FOR«-Anweisung ein weiterer »FOR«-Befehl stehen, innerhalb diesem ein weiterer, und so weiter. Das folgende Beispiel soll dies schematisch darstellen:

```
FOR loop1 := a1 TO e1 DO
    FOR loop2 := a2 TO e2 DO
        FOR loop3 := a3 TO e3 DO ...
```

Die Schachtelungstiefe, das heißt die Anzahl der Schleifen, die man ineinanderpacken kann, hängt von der Leistungsfähigkeit jedes Pascal-Compilers ab, da er sich ja alle Schachtelungen merken muß. Listing 4 zeigt ein kleines Programm, welches das kleine Einmaleins ausdruckt und dabei zwei FOR-Schleifen verwendet.

```
PROGRAM schleife1;
VAR index: integer;
    BEGIN
      FOR index := 1 TO 10 DO BEGIN
          WRITE('Das Quadrat von ',index,' ist ');
          WRITE(SQR(index));
      END;
    END.
```

Listing 3. Einfache FOR-Schleife

```
PROGRAMM schleife2;
VAR index1,index2: integer;
BEGIN
  FOR index1 := 1 TO 10 DO BEGIN
    FOR index2 := 1 TO 10 DO BEGIN
      Write(index1,'*',index2,'=',index1*index2);
    END;
  END;
END.
```

Listing 4. Verschachtelte FOR-Schleife

```
PROGRAM raufundrunter;
VAR index: integer;
    BEGIN
      WRITELN('Ich zähle jetzt rauf!');
      FOR index := 1 TO 20 DO WRITELN(index);
      WRITELN('Ich zähle jetzt runter!');
      FOR index := 20 DOWNTO 1 DO WRITELN(index);
    END.
```

Listing 5. FOR..DOWNTO

Die Möglichkeit der Verschachtelung ist natürlich nicht nur bei »FOR«, sondern auch bei all den weiteren Schleifenarten von Pascal vorhanden.

Die normale »FOR«...TO-Anweisung zählt die Kontrollvariable nach der festgelegten Ordnung des jeweiligen Datentyps nach oben. Eine Variante dazu ist die »FOR...DOWNTO«-Schleife. Sie zählt vom Anfangswert angefangen in der Ordnung abwärts. Der Aufbau ist der »FOR...TO«-Schleife ähnlich:

```
FOR (Kontrollvariable) := (a) DOWNTO (e)
    DO (Anweisung);
```

Die »FOR...DOWNTO«-Anweisung arbeitet nach dem gleichen Prinzip wie der »FOR...TO«-Befehl und besitzt auch dessen Möglichkeiten der Verschachtelung und der Benutzung von Verbundanweisungen. Listing 5 demonstriert den Unterschied zwischen »FOR...TO« und »FOR...DOWNTO«.

Der nächste Wert einer FOR-Schleife ist bei »FOR...TO« immer der unmittelbare Nachfolger und bei »FOR... DOWNTO« der unmittelbare Vorgänger des aktuellen Zählerwertes. Die Schrittweite ist also immer auf den Wert 1 oder entsprechend -1 festgelegt. Andere Schrittweiten sieht Pascal nicht vor. Es liegt dann durch besondere Rechnungen an den Künsten des Programmierers, auch andere Schrittweiten zu erzeugen. Mit Hilfe einer zweiten Variablen, die quasi als fiktive Kontrollvariable fungiert, kann man eine Schrittweite von 2 zum Beispiel folgendermaßen erzeugen:

```
FOR x := 0 TO (z - 1) DIV 2 DO BEGIN
    y := x * 2 + 1; (* eigentliche Schleife *)
END;
```

»y« nimmt hierbei die Werte 1,3,5,7,9,...z an. Entsprechend können auch andere Schrittweiten erreicht werden. Bei dem Datentyp char zum Beispiel dürfte dies etwas schwieriger zu gestalten sein.

Neben den beiden »FOR«-Varianten gibt es noch andere Schleifenarten, die erst unter bestimmten Bedingungen ausgeführt oder beendet werden. Dafür ist aber das Verständnis des Begriffs »Bedingungen« notwendig. Mit der Erklärung von Bedingungen tauchen wir auch etwas in die boolsche (logische) Arithmetik ein. Jetzt wird auch der Begriff »relationale Operatoren«, der schon am Anfang dieses Kurses bei der Behandlung von Datentypen erwähnt wurde, benötigt. In Bild 8 sind alle diese besonderen Operatoren mit ihren Bedeutungen aufgelistet. Diese, im Deutschen auch Vergleichsoperatoren genannten Zeichen, können unter Beach-

Operator	Bedeutung
=	ist gleich
<>	ist ungleich
<	ist kleiner als
>	ist größer als
<=	ist kleiner oder gleich als
>=	ist größer oder gleich als
TRUNC(x)	ganzzahliger Wert von x
ROUND(x)	Aufrunden von x

Bild 8. Relationale Operatoren

tung einiger Regeln begrenzt mit allen Standarddatentypen verknüpft werden. Allgemein geschrieben sieht ein Vergleichsausdruck so aus:

(Ausdruck1) (Operator) (Ausdruck2)

Für »Ausdruck1« und »Ausdruck2« können beliebige Ausdrücke jeden Datentyps stehen. Wichtig ist nur, daß sie untereinander vom gleichen Typ sind. Ein Ausdruck mit Vergleichsoperatoren liefert immer ein Ergebnis des Typs boolean, das heißt »true« oder »false« für wahr oder falsch. Hier ein Beispiel eines solchen Ausdrucks:

5 + 6 <> 7 * 2 - 3

Wie das Ergebnis dieses Ausdruckes ausfällt, ist leicht festzustellen. 5 plus 6 ergibt 11. Die rechte Seite hat ebenfalls das Ergebnis 11, da 7 mal 2 gleich 14 und das minus 3 gleich 11 ist. Der Operator heißt aber »ungleich«. Rechts und links stehen aber die gleichen Ergebnisse, womit die Beziehung falsch wäre und das Resultat »false« lautet. Zum besseren Verständnis einige weitere Beispiele:

9 = 5 + 4	ergibt »true« (die Aussage stimmt)
4 < 7	ergibt »true«
7 > 10	ergibt »false«
5 + 2 <> 6	
+ 3	ergibt »true«
10 <= 9	ergibt »false«

Solche Vergleichsausdrücke können auch als Bedingungen interpretiert werden, die erfüllt sind, wenn das Ergebnis »true« ist, oder nicht erfüllt sind, sollte es »false« sein. Allgemein kann man alle Werte des Typs boolean als Bedingungen verwenden. Auch einfache Variablen des Typs boolean sind damit gemeint. Eine Variable beispielsweise mit Namen »fertig« (natürlich muß sie vorher als Datentyp boolean deklariert sein), darf durchaus als eine Bedingung benutzt werden, die erfüllt ist, wenn sie den Wert »true« hat, und nicht erfüllt ist, sollte ihr Wert »false« sein. Mehrere Bedingungen können wiederum zusammengefaßt werden. Da sie vom Typ boolean sind, sind Verknüpfungen mit den logischen Operatoren »AND«, »OR« und »NOT« erlaubt. Eine Verknüpfung mit »AND« wird nur wahr, das heißt erhält das Ergebnis »true«, wenn beide Teilbedingungen selbst wahr sind:

```
(27 = 9 * 3) AND (5 <= 6)
```

hat das Ergebnis »true« und ist deshalb als Gesamtbedingung wahr. Die Teilbedingungen müssen bei Verwendung der drei booleschen Operatoren geklammert werden, da »AND«, »OR« und »NOT« mathematisch stärker binden als die Vergleichsoperatoren. Ohne Klammer würde bei unserem Beispiel der Operator »AND« nur für die direkt benachbarten Werte, also 3 und 5, gelten.

```
27 = 9 * 3 AND 5 <= 6
```

Wir haben aber gelernt, daß boolsche Operatoren nur in Zusammenhang mit boolschen Werten verwendet werden dürfen. 3 und 5 sind aber Integer-Werte, was eine Fehlermeldung zur Folge hätte. Bei »OR« reicht es bereits, wenn eine der beiden Bedingungen wahr ist, um selbst als Verbundbedingung wahr zu werden.

```
(9 < 7) OR (15 + 5 = 20)
```

Die erste Bedingung in diesem Beispiel ist falsch, da neun nicht kleiner als sieben ist. Sie erhält also den Wert »false«. Die zweite Bedingung dagegen ist wahr (15 plus 5 ist in der Tat 20) und wird daher »true«, womit der Gesamtausdruck »true« wird. Es ist ja mindestens eine Bedingung (in diesem Fall die zweite) wahr. »NOT«, die logische Negation, benötigt, wie auch in Bild 6 erkennbar, nur einen Operanden beziehungsweise eine Bedingung, wie zum Beispiel

```
NOT(6 <= 2 * 3)
```

Die Bedingung in Klammern erhält den Wert »true«, da zwei mal drei sechs ergibt. »NOT true« hat das Ergebnis »false«, womit die Gesamtbedingung den Wert »false« ergibt. Damit Sie testen können, ob Sie mit den Vergleichoperatoren und den logischen Verknüpfungen vertraut sind, können Sie ja einmal folgende zusammengesetzte Bedingung daraufhin prüfen, ob sie erfüllt ist oder nicht:

```
((7 < (4 * 3)) + 9 <> 17) AND
((5 - 2) > ((4 MOD 2) = (6 DIV 3)))
```

Die Auflösung erfahren Sie am Ende des Kurses. Man stellt sich nun die Frage, was man mit diesen Bedingungen alles anfangen kann. Neben den Zählschleifen »FOR...TO« und »FOR...DOWNTO« hält Pascal einige andere sogenannnte bedingte Schleifen bereit.

Im Gegensatz zu »FOR...DO«, das mindestens einmal ausgeführt wird, sobald der Computer darauf trifft, wird eine »WHILE...DO«-Schleife nur beachtet, wenn eine bestimmte Bedingung erfüllt ist. Allgemein wird eine »WHILE...DO«-Anweisung wie folgt formuliert:

```
WHILE (Bedingung) DO (Anweisung);
```

Als Bedingung kann jeder Ausdruck gelten, dessen Ergebnis nur »true« (wahr) oder »false« (falsch) werden kann. Wir haben diese ja soeben behandelt. Hinter dem Wort »DO« steht wieder die Anweisung, die in der Schleife ausgeführt werden soll. Benötigt man mehrere Befehle in der Schleife, so müssen diese, wie Sie schon wissen, mit »BEGIN« und »END« zu einer Verbundanweisung zusammengefaßt werden:

```
WHILE (Bedingung) DO    BEGIN (Anweisungen) END;
```

Ist die Bedingung hinter »WHILE« erfüllt, wird die Anweisung hinter »DO« ausgeführt. Nach Beendigung eines Durchgangs wird die Bedingung nochmals geprüft. Die Schleife wird ein weiteres Mal durchlaufen, wenn die Bedingung noch wahr ist. Sollte sie den Wert »false« ergeben, wird die »WHILE...DO«-Schleife abgebrochen. Für den Fall, daß die Bedingung von Anfang an falsch ist, werden die Anweisungen der Schleife einfach ignoriert und im Programm fortgefahren. Ein Beispiel für die Funktionsweise sei folgendes Programm in Listing 6.

Wenn Sie das Programm betrachten, können Sie feststellen, daß anders als bei der Kontrollvariablen der FOR-Schleife die verwendeten Variablen der Bedingung auch innerhalb der Schleife selbst verändert werden dürfen. So kann ein Ergebnis, das während des Schleifenablaufs berechnet wird, einen weiteren Durchlauf verhindern, wenn dadurch die Bedingung nicht mehr erfüllt wird. Listing 6 zeigt dies auf einfache Weise. Selbstverständlich sind auch bei »WHILE...DO« Verschachtelungen gestattet, wie es das folgende Schema zeigt:

```
WHILE (Bedingung 1) DO
      WHILE (Bedingung 2) DO
            WHILE (Bedingung 3) DO ...
```

Ist (Bedingung1) erfüllt, wird der Befehl hinter »DO« abgearbeitet. Dies ist eine weitere »WHILE...DO«-Anweisung, worauf (Bedingung2) auf ihre Gültigkeit untersucht wird. Sollte auch diese »true« sein, so wird noch eine weitere Schachtelungsebene hinabgestiegen und so weiter.

Im Gegensatz zu »WHILE...DO« steht bei der Schleifenart, die jetzt erklärt werden soll, das Kriterium zum Ablauf der Schleife an deren Ende. Sie hat die allgemeine Form

```
REPEAT (Anweisung) UNTIL (Bedingung);
```

```
PROGRAM WEDELN;
    VAR J:INTEGER;
    BEGIN
      J:=1;
      WHILE J<=3 DO
        BEGIN
          WRITELN('   *');
          WRITELN('    *');
          WRITELN('     *');
          WRITELN('    *');
          J:=J+1
         END;
       END.
```

Listing 6. WHILE..DO

In Klartext übersetzt etwa: »Wiederhole die Anweisungen, bis die Bedingung erfüllt ist«. Dieses Mal bestimmt die Bedingung, ob die Schleife abgebrochen wird. Bei »WHILE...DO« dagegen entschied die Bedingung, ob die Schleife überhaupt begonnen wurde. Zwischen »REPEAT« und »UNTIL« stehen die Anweisungen, die in der Schleife ausgeführt werden sollen. Es können beliebig viele Anweisungen sein, die aber nicht, wie bei den anderen Schleifenarten, vorher mit »BEGIN« und »END« zu einer Verbundanweisung zusammengefügt werden müssen. Die beiden Schlüsselwörter »REPEAT« und »UNTIL« übernehmen dies praktisch schon von selbst. Anders als bei »WHILE...DO« wird die Bedingung erst am Ende eines Schleifendurchlaufs kontrolliert. Ist die Bedingung wahr, wird die Schleife abgebrochen und im Programm fortgefahren. Da das Abbruchkriterium zum Schluß steht, wird die »REPEAT...UNTIL«-Schleife stets mindestens einmal durchlaufen. Selbst wenn die Bedingung hinter »UNTIL« schon anfänglich nicht erfüllt ist, wird die Schleife einmal komplett abgearbeitet. Die Bedingung wird ja erst am Ende der Schleife geprüft. Listing 7 zeigt diesen Sachverhalt an einem kleinen Beispiel.

Auch »REPEAT...UNTIL«-Schleifen lassen sich ineinanderschachteln, wie das folgende Schema zeigt:

```
REPEAT
    REPEAT
        REPEAT
            (Anweisungen)
        UNTIL (Bedingung 3);
    UNTIL (Bedingung 2);
UNTIL (Bedingung 1);
```

Wie bei den anderen Schleifentypen, wird immer die innerste Schleife als erstes abgearbeitet. Sollte in unserem Schema zum Beispiel die innerste Schleife eine Endlosschleife sein (das heißt eine Schleife, die sich endlos wiederholt, da die Bedingung 3 nie erfüllt wird) ist es zwecklos, wenn die Bedingungen 2 und 1 längst erfüllt sind und zu einem Abbruch führen würden.

Damit hätten wir alle in Pascal möglichen Schleifenkonstruktionen abgehandelt. Sie gehören mit zu den wichtigsten Befehlen, die den Ablauf eines Pascal-Programmes steuern. Man darf aber eine weitere Steuerform von Pascal nicht vernachlässigen.

Es gibt in Pascal, ähnlich wie bei anderen Programmiersprachen, eine Befehlsfolge, die ermöglicht, nur dann eine Anweisung auszuführen, wenn eine bestimmte Bedingung erfüllt ist. Man bezeichnet dies als »bedingte Verzweigung«. Sie hat folgenden Syntax:

```
IF (Bedingung) THEN (Anweisung);
```

Die bedingte Verzweigung ähnelt dem IF-THEN-Befehl in Basic und hat im Prinzip auch die gleiche Bedeutung. Die Anweisung hinter »THEN« wird nur dann ausgeführt, wenn die Bedingung nach »IF« erfüllt ist, also den Wert »true« hat. Eine korrekte »IF«-Anweisung wäre zum Beispiel:

```
IF betrag <= 0 THEN WRITE('Du hast kein
Geld mehr.');
```

Wird die Variable »betrag« kleiner oder gleich Null und die Bedingung »betrag <= 0« somit wahr, wird die »WRITE«-Anweisung hinter »THEN« ausgeführt. Der Ausdruck lautet dann: »Du hast kein Geld mehr«.

Hat die Bedingung den Wert »false«, wird die Anweisung ignoriert. Mehrere zusammengehörige Anweisungen werden wieder zu einer Verbundanweisung zusammengefaßt. Dies geschieht, wie Sie schon wissen, mit »BEGIN« und »END«.

```
IF (Bedingung) THEN
    BEGIN (Anweisungen)
    END;
```

Mit dem »IF«-Befehl können wir nun im Pascal-Programm zum Beispiel Eingaben von der Tastatur auf ihre Richtigkeit überprüfen, wie es Listing 8 zeigt. Die Eingabe der Variable »betrag« soll nur positive Zahlen gestatten. Bei Eingabe eines negativen Wertes soll sich das Programm beschweren.

Bedingungen, wie wir sie kennen, haben immer eine eindeutige Lösung, das heißt entweder wahr (»true«) oder falsch (»false«). In Listing 8 zeigt sich dies recht deutlich. Die Variable »betrag« kann entweder nur positiv oder negativ sein. Die beiden »IF«-Anweisungen berücksichtigen diese Fälle. Eine »IF«-Verzweigung beschäftigt sich mit dem Fall, daß »betrag« größer oder gleich Null ist, das andere »IF«-Statement mit der Bedingung »betrag« kleiner Null. Zur Vereinfachung solcher Sachverhalte bietet Pascal eine Erweiterung des »IF«-Befehls durch Hinzufügen des Schlüsselwortes »ELSE«.

```
IF (Bedingung) THEN (Anweisung 1)
               ELSE (Anweisung 2);
```

»ELSE« ermöglicht das Berücksichtigen beider Lösungsfälle einer Bedingung. Ist die Bedingung wahr, so wird nur Anweisung 1 ausgeführt. Sollte das Ergebnis der Bedingung jedoch »false« sein, wird Anweisung 2 hinter »ELSE« abgearbeitet. Damit kann das Programm in Listing 8 wesentlich einfacher formuliert werden. Listing 9 zeigt das modifizierte Programm mit der »IF...THEN...ELSE«-Verzweigung.

Wie bei Pascal-Schleifen sind hinter »THEN« und »ELSE« auch Verbundanweisungen möglich. Sie müssen wie immer jeweils von »BEGIN« und »END« umgeben sein. Folgendes Schema verdeutlicht dies:

```
IF (Bedingung) THEN BEGIN
                    (Anweisungen)
                    END
               ELSE BEGIN
                    (Anweisungen)
                    END;
```

```
PROGRAM NOTEN3;
  VAR SCHUELERNUMMER,NOTE,ZAEHLER,SUMME:INTEGER;
  BEGIN
    SUMME:=0;
    ZAEHLER:=0;
    WRITELN(' SCHUELER','NOTE':6);
    READLN(SCHUELERNUMMER,NOTE);
    REPEAT
        SUMME:=SUMME+NOTE;
        ZAEHLER:=ZAEHLER+1;
        READLN(SCHUELERNUMMER,NOTE);
    UNTIL SCHUELERNUMMER=0;
    WRITELN(' DURCHSCHNITT=',ROUND(SUMME/ZAEHLER));
    WRITELN(' ZAEHLER='ZAEHLER);
  END.
```

Listing 7. REPEAT..UNTIL

```
PROGRAM betrag;
VAR betrag: integer;
  BEGIN
    READ(betrag)
    IF betrag < 0 THEN WRITE('Falsche Eingabe!');
    IF betrag >= 0 THEN WRITE('Der Betrag ist'
    ,betrag,' DM.');
  END.
```

Listing 8. IF..THEN

```
PROGRAM betrag;
VAR betrag: integer;
  BEGIN
    READ(betrag);
    IF betrag < 0 THEN WRITE('Falsche Eingabe!')
                  ELSE WRITE('Der Betrag ist ',betrag,
                        ' DM.');
  END.
```

Listing 9. IF..THEN..ELSE

Sollten Sie einmal bei zusammengehörenden Anweisungen die Schlüsselworte »BEGIN« und »END« vergessen, kann dies zu einem vollkommen anderen Programmablauf oder gar einer Fehlermeldung führen.

```
IF (Bedingung) THEN (Anweisung 1);
                    (Anweisung 2);
```

ist zum Beispiel keinesfalls das gleiche wie

```
IF (Bedingung) THEN BEGIN
                    (Anweisung 1);
                    (Anweisung 2)
                    END;
```

Im ersten Fall gehört die Anweisung 2 nicht mehr zum »IF...THEN«-Statement und wird immer ausgeführt. Im zweiten Fall dagegen wurden die beiden Anweisungen mit »BEGIN« und »END« zu einer Verbundanweisung zusammengeschlossen und gehören nun beide zur »IF«-Bedingung. Die Anweisung

```
IF (Bedingung) THEN (Anweisung 1);
                    (Anweisung 2)
               ELSE (Anweisung 3);
```

ist sogar syntaktisch falsch und würde dem Pascal-Compiler nicht gefallen. Warum? Da Anweisung 2 mit Anweisung 1 nicht zu einer Verbundanweisung zusammengefaßt wurde, wird sie als allgemeiner Befehl aufgefaßt. Für den Compiler ist der »IF«-Befehl also bereits nach Anweisung 1 beendet. Das »ELSE« wird nun nicht mehr als zugehörig erkannt, da zwischen »THEN« und »ELSE« die allgemeine Anweisung, nämlich Anweisung 2, steht. Ein »ELSE« ohne »IF...THEN« kennt der Compiler aber nicht, was zu einer Fehlermeldung führt. Die Behandlung von Verbundanweisungen sollte deshalb mit viel Sorgfalt geschehen, sonst werden ihnen schlimme Programmierfehler, die nicht gerade leicht zu finden sind, das Leben schwer machen. Muß unter mehr als zwei Alternativen ausgewählt werden, bietet sich die Möglichkeit, unmittelbar nach »THEN« oder »ELSE« einen weiteren »IF«-Befehl zu verwenden, das heißt mehrere »IF«-Anweisungen ineinanderzuschachteln. Das Schema zur Verschachtelung von »IF«-Verzweigungen ist dem der verschachtelten Schleifen sehr ähnlich.

```
IF (Bedingung 1) THEN (Anweisung 1)
                 ELSE
IF (Bedingung 2) THEN (Anweisung 2)
                 ELSE
IF (Bedingung 3) THEN (Anweisung 3)
                 ELSE ...
```

Analog dazu kann die Schachtelung auch hinter »THEN« geschehen. Man sollte auf allzuviele »IF«-Befehle hintereinander lieber verzichten, da ab einer gewissen Schachtelungstiefe die Übersicht stark leiden kann. Listing 10 zeigt ein einfaches Programm, welches die Verschachtelung von »IF«-Anweisungen aufzeigt.

»IF...THEN...ELSE« war nun die letzte der in Pascal verfügbaren Strukturanweisungen. Jetzt ist es an der Zeit, daß Sie auch die Möglichkeit kennenlernen, eigene Datentypen zu erstellen. Bei der Besprechung der Standardtypen wurden sie bereits erwähnt.

In manchen Fällen ist es angebracht, Variablen zu verwenden, die nur eine gewisse Auswahl an Zahlen, Zeichen oder nur bestimmte Zeichenketten beinhalten können. Wenn Sie zum Beispiel ein Würfelspiel für vier Spieler programmieren wollen, so wäre es vorteilhaft, die vier Spielfarben, beispielsweise Rot, Gelb, Grün und Blau, in einer Variablen zu definieren, die nur diese Werte annehmen kann. Pascal ermöglicht die Erstellung solcher und ähnlicher Datentypen durch das Schlüsselwort »TYPE«. Die Definition eigener Datentypen erfolgt im Deklarationsteil eines Pascal-Programms. Sie muß zwischen den Konstanten- und Variablendeklarationen geschehen, da sie zur Variablendefinition gebraucht werden. Die Syntax von »TYPE« hat folgendes Format:

```
TYPE (Name) = (Typ);
```

Wie bei »CONST« und »VAR« können beliebig viele Definitionen gemacht werden. Das Wort »TYPE« muß dabei nur am Anfang der gesamten Typendeklaration stehen. Mit »TYPE« wird einem eigens erstellten Datentyp ein Name gegeben, der dann in den Variablendeklarationen wie die Namen der Standardtypen integer, real, char, boolean und string verwendet werden kann. Rechts neben dem Gleichheitszeichen wird angegeben, wie der Typ auszusehen hat. Diese Angabe kann bei fortgeschrittener Programmierung, wie zum Beispiel bei Arrays und Records, relativ kompliziert werden. Diese sogenannten »strukturierten Datentypen« sollen uns aber im Moment nicht beschäftigen. Wenden wir uns lieber den einfacheren Typen zu, um die Bedeutung von »TYPE« genau zu verstehen. Es gibt in Pascal zwei Möglichkeiten, mit »TYPE« einfache Datentypen zu erzeugen. Dies sind die Aufzählungstypen, im Englischen auch »enumerated types« genannt, und die Ausschnittstypen (englisch »subrange types«). Damit erweitert sich der Vorrat an Datentypen neben den Standardtypen auf »enumerated«- und »subrange«-Typen.

Der Aufzählungstyp ist gleichzeitig die einfachste Methode, einen neuen Datentyp zu erzeugen. Wie es der Begriff schon ausdrückt, geschieht dies durch genaue Aufzählung der Werte, die der Datentyp annehmen darf. Die Werte dürfen nur aus Namen bestehen, die man als Konstanten, des definierten Typs bezeichnen kann. Zahlen können vom Compiler nicht akzeptiert werden. Man erzeugt einen Aufzählungstyp, indem man die Werte durch Kommata getrennt hinter dem Gleichheitszeichen niederschreibt. Sie müssen von Klammern umgeben sein. Zum Abschluß steht wie üblich das Semikolon. Um das Beispiel des Würfelspiels wieder aufzugreifen, wäre eine Typendeklaration, die die vier Spielfarben beinhaltet, folgende:

```
TYPE farbe = (rot,gelb,gruen,blau);
```

Damit haben wir einen neuen Datentyp erzeugt, der den Namen »farbe« hat und nur die Werte rot, gelb, gruen und blau annehmen kann. Den Namen »farbe« kann man nun in der Variablendefinition verwenden, wie zum Beispiel

```
VAR sieger: farbe;
```

Die Variable »sieger« hat den vorher definierten Typ »farbe« erhalten und darf von jetzt an nur die vorbestimmten Werte

```
PROGRAM WAHLEN;
  CONST KONSERVATIVE=1;
    RADIKALE=2;
    UNABHAENGIGE=3;
  VAR STIMME,RECHTS,LINKS,MITTE,ZAEHLER:INTEGER;
  BEGIN
    RECHTS:=0;
    LINKS:=0;
    MITTE:=0;
    WRITTELN('STIMMEN EINGEBEN');
    READLN(STIMME);
  WHILE STIMME<>-1 DO
    BEGIN
      IF STIMME=KONSERVATIVE THEN
         RECHTS:=RECHTS+1
      ELSE
         IF STIMME=RADIKALE THEN
            LINKS:=LINKS+1
         ELSE
            IF STIMME=UNABHAENGIG THEN
               MITTE:=MITTE+1;
      READLN(STIMME);
    END;
  ZAEHLER:=RECHTS+LINKS+MITTE;
  WRITELN('KONSERV.':10,'RADIKAL':10,
          'UNABH.':10,'GESAMT':10);
  WRITELN(RECHTS:10,LINKS:10,MITTE:10,ZAEHLER:10);
END.
```

Listing 10. IF-Verschachtelung

annehmen. Dies geschieht auf die übliche Art und Weise:

```
sieger := gruen;
```

Der Name »gruen« ist dabei keine andere Variable, sondern eine Konstante des Datentyps »farbe«, den wir zuvor bestimmt hatten. Wie Sie sehen können, werden diese Konstanten wie ganz normale Werte anderer Typen (beispielsweise Integer-Zahlen) behandelt. So können mit diesen Werten auch Bedingungen gebildet werden, wie zum Beispiel

```
sieger := rot
```

Eine »IF«-Anweisung würde wie folgt aussehen:

```
IF sieger = rot THEN WRITE('Spieler Rot
ist Sieger.');
```

Diese einfache Handhabung kann sehr nützlich sein. So haben die Werte des Aufzählungstyps durch das Aufschreiben in einer gewissen Reihenfolge eine Ordnung erhalten, die es erlaubt, die Funktionen »succ«, »pred« und »ord« zu verwenden. Diese drei Funktionen arbeiten dabei genauso wie bei den Standarddatentypen. Die Ordnung der Spielfarben würde folgendermaßen lauten:

Wert	rot	gelb	gruen	blau
Ordnungszahl	0	1	2	3

Unter der Voraussetzung unserer oben gewählten Definition und der Ordnungszahlen des Spielfarbentyps »farbe«, sollen einige Beispiele die Funktionen »succ()«, »pred()« und »ord()« im Zusammenhang mit selbstdefinierten Datentypen erklären:

succ(rot)	ergibt den Wert gelb
succ(gruen)	ergibt den Wert blau
succ(blau)	ist nicht definiert
pred(blau)	ergibt den Wert gruen
pred(gelb)	ergibt den Wert rot
ord(rot)	ergibt den Wert 0
ord(gruen)	ergibt den Wert 2

Die Operatoren der Standardtypen kann man allerdings nicht anwenden, was auch logisch ist. Was sollte die Rechnung »rot * gruen« schon ergeben? Die Vergleichsoperatoren sind dagegen sinnvoller. Durch sie ist, wie bereits erwähnt, die Erstellung von Bedingungen möglich. Da eine Ordnung in der Reihe der Werte besteht, sind folgende Ausdrücke korrekt und haben eine Lösung:

rot >= gruen	ergibt »false«
blau <> gelb	ergibt »true«
gruen < gelb	ergibt »false«

Der neue Datentyp kann auch als Kontrollvariable einer »FOR...TO«-Schleife fungieren. Die einzelnen Werte werden dabei der Reihe nach vom Anfangs- bis zum Endwert durchgezählt. Ein Beispiel, das alle Werte unseres Farbenbeispiels durchläuft, wäre:

```
FOR sieger := blau DOWNTO rot DO (Anweisung);
```

Ebenso nützlich wäre es, wenn die Anweisung

```
WRITE(sieger);
```

den augenblicklichen Wert der Variablen »sieger« ausgeben würde. Dies ist in Pascal jedoch nicht vorgesehen. Man muß sich dadurch mit einigen »IF«-Anweisungen behelfen.

```
PROGRAM spielfarben;
TYPE farben = (rot,gelb,gruen,blau);
VAR spieler: farben;
    eingabe: string;
    BEGIN
      READ(eingabe);
      IF eingabe = 'rot' THEN spieler := rot
         ELSE IF eingabe = 'gelb' THEN spieler := gelb
           ELSE IF eingabe = 'gruen' THEN spieler := gruen
             ELSE IF eingabe = 'blau' THEN spieler := blau
               ELSE WRITE('Unbekannte Farbe');
  END.
```

Listing 11. Eingabe von Aufzählungstypen

```
WRITE('Der Sieger ist Spieler ');
 IF sieger = rot THEN WRITE('Rot')
    ELSE IF sieger = gelb THEN WRITE('Gelb')
       ELSE IF sieger = gruen THEN WRITE('Gruen')
          ELSE WRITE('Blau');
```

Die Eingabe solcher Aufzählungskonstanten im richtigen Wortlaut bringt jedoch in Standard-Pascal noch größere Schwierigkeiten mit sich. Mit der normalen »READ«-Anweisung ist eine direkte Eingabe nämlich nicht möglich. Hier wäre es angebracht, eine eigene Einleseroutine zu schreiben, die auch Daten von selbstdefinierten Datentypen akzeptiert. Die Programmierung dürfte für einen Anfänger zu kompliziert werden, so daß man sich mit einer einfacheren, aber um so schreibaufwendigeren Methode begnügen muß. Für den UCSD-Pascal-Anwender ist dies sogar noch etwas leichter zu gestalten, da er den Standardvariablentyp string zur Verfügung hat. Mit ihm läßt sich eine Eingabe von Daten ähnlich wie die Ausgabe gestalten. Listing 11 zeigt eine Lösung des Problems mit Hilfe des UCSD-Datentyps string.

Eine andere Art von selbstdefinierten Typen sind die Ausschnittstypen (»subrange types«). Sie berufen sich auf bereits vorhandene Typen, wie zum Beispiel die Standarddatentypen oder einen selbstdefinierten Aufzählungstyp. Nicht verwenden darf man den Datentyp real und string. Auch hier gibt der Name (Ausschnittstyp) eine Erklärung seiner selbst. Aus definierten Werten, wie beispielsweise Integer-Zahlen, wird ein gewisser Ausschnitt gewählt, der dann den neuen Typ darstellt. Seine Werte dürfen sich nur im Rahmen dieses Ausschnitts bewegen. Das Zeichen, das die Auswahl eines Ausschnitts ermöglicht, besteht aus zwei Punkten (..), dem »compound symbol« und zwar nach folgendem Muster:

```
(Konstante 1) .. (Konstante 2)
```

»Konstante 1« und »Konstante 2« sind hierbei Werte eines bereits definierten Typs, wobei »Konstante 1« in der Ordnung weiter vorne liegen muß, das heißt eine niedrigere Ordnungszahl haben muß als »Konstante 2«. Den Datentyp, aus dem der Ausschnitt gewählt wird, nennt man auch den Grundtyp oder in Englisch »host type« des Ausschnittstyps. Der neu definierte Typ verhält sich genauso wie sein Grundtyp. Das heißt, es lassen sich alle im Grundtyp definierten Operatoren und Funktionen anwenden. Seinen eingeschränkten Wertebereich darf er dabei jedoch nicht verlassen. Hier einige Beispiele eines »subrange« types:

```
TYPE jahr = (jan,feb,mar,apr,mai,jun,jul,aug,sep,
okt,nov,dez);
      sommer = jun..sep;
      buchstaben = 'a'...'z';
      ausgaben = 0..15000;
```

Betrachten wir den neuen Typ »sommer«, so erkennen wir, daß auch ein Aufzählungstyp Grundtyp für einen Ausschnittstyp sein kann. Wie beim »enumerated type« kann der neu definierte Typ Datentyp für Variablen werden.

```
VAR ferien: sommer;
    monat: jahr;
    tasten: buchstaben;
```

Es ist dabei unbedingt zu beachten, daß während des Programmablaufs die Bereiche der Ausschnittstypen eingehalten werden, da es ansonsten zu unangenehmen Fehlern kommt. Die Schleife unten zum Beispiel wird mit Sicherheit eine Fehlermeldung erzeugen. Überlegen Sie, warum.

```
FOR monat := jun TO okt DO
    ferien := succ(ferien);
```

Gerade bei Datentypen mit vielen Werten, die wie bei dem »enumerated type« »jahr« mit »WRITE« nicht direkt ausgegeben werden können, stellt man sich die Frage, ob Pascal wirklich so unvollkommen ist, daß eine Auswahl von verschiedenen Werten nur durch verschachtelte »IF«-Anweisungen vorgenommen werden kann. Wir haben dies ja bereits zu einem früheren Zeitpunkt besprochen. Pascal hat dafür natürlich

vorgesorgt. Die »CASE«-Anweisung ermöglicht eine gezielte Fallunterscheidung von mehreren sich ausschließenden Werten einer Variablen. Sie tritt in folgender Form auf:

```
CASE (Ausdruck) OF (case-Konstantenliste) END;
```

Der Ausdruck ist in diesem Fall der Name der Variablen, an der die Fallunterscheidung geschehen soll. Man nennt ihn deshalb auch Selektor. Selektor kann jede Variable der uns bekannten Datentypen sein, mit Ausnahme des Typs real. Nach dem Wort »OF« beginnt die Liste der Konstanten, in die unterschieden werden soll. Sie müssen selbstverständlich vom Typ des Selektors sein. Die Reihenfolge der einzelnen Konstanten ist frei wählbar. Eine Konstante darf dabei aber nur einmal in dieser Liste vorkommen, da ja sonst nicht eindeutig unterschieden werden kann. Hinter jeder Konstanten steht schließlich, mit einem Doppelpunkt getrennt, die Anweisung, die ausgeführt werden soll, wenn die Variable den Wert der Konstanten annimmt. Die Liste muß mit »END;« abgeschlossen werden. Greifen wir nochmals das Beispiel mit dem Würfelspiel auf und formulieren eine »CASE«-Anweisung mit dem Selektor »spieler«, der ja bekanntlich eine Variable des Aufzählungstyps ist und nur die Werte rot, gelb, gruen und blau annehmen kann:

```
CASE spieler OF
        rot: WRITE('Rot');
       gelb: WRITE('Gelb');
      gruen: WRITE('Grün');
       blau: WRITE('Blau');
END;
```

Je nachdem, welchen Wert die Variable »spieler« im Moment hat, wird die entsprechende »WRITE«-Anweisung ausgeführt.

Eine Bedingung für die »CASE«-Anweisung ist, daß die Variable eine der aufgelisteten Konstanten annehmen muß, bevor der »CASE«-Ausdruck abgearbeitet wird. Findet der Computer den Wert des Selektors in der Liste nicht, so wird in Standard-Pascal das Programm mit einer Fehlermeldung unterbrochen. UCSD-Pascal dagegen führt dann immer die Anweisungen hinter der ersten Konstante aus und wirkt somit wie eine Nullanweisung. Eine »CASE«-Anweisung ohne die »CASE«-Konstantenliste ist nicht definiert und wird bei den meisten Rechenanlagen mit einem Fehler quittiert. Die beiden neuen Datentypen (enumerated und subrange type) sind in Verbindung mit dem »CASE«-Befehl ein schönes Hilfsmittel zur Bewältigung von besonderen Problemen.

Sollen jedoch sehr viele gleichartige Daten verarbeitet werden, so werden wir mit den einfachen Datentypen, die wir schon kennen, bald an Grenzen stoßen. Für eine Bearbeitung von 10 000 Daten würde ein Programm, das mit den bisher besprochenen Mitteln geschrieben ist, viel zu lang werden. Man bedenke auch, daß schon das Erfinden von 10 000 Variablennamen unmöglich wäre von den Anweisungen, die diese Variablen verarbeiten, ganz zu schweigen. Kommen wir deshalb zu den sogenannten »strukturierten Datentypen«, die uns bei der Bewältigung von solchen Problemen von sehr großem Nutzen sind.

Hat man sehr viele Daten des gleichen Typs, so können diese in Pascal zu einem »Feld« (englisch: array) zusammengefaßt werden, welches mit nur einem einzigen Namen aufgerufen werden kann. Jeder Wert in diesem Feld bekommt eine Nummer in aufsteigender Ordnung, unter der er aufgerufen werden kann. Diese Nummer wird auch Index genannt. Man nennt Felder, deren Elemente nur einen Index haben, auch »eindimensionale Arrays«. Mit Hilfe des Schlüsselwortes »ARRAY« kann ein solches Feld erstellt werden. Allgemein wird dies so formuliert:

```
ARRAY [(Indextyp)] OF (Komponententyp)
```

Der Indextyp stellt eine endliche Anzahl von Werten dar, die den Indexbereich des Arrays bestimmen. Er muß ein Aufzählungs- oder Ausschnittstyp sein oder einer der Typen boolean und char. Der Komponententyp dagegen kann beliebiger Art sein, insbesondere auch ein selbstdefinierter oder strukturierter Typ, wie zum Beispiel wiederum ein Arraytyp (dieser Fall wird später noch erläutert). Er bestimmt den Datentyp des gesamten Feldes (ein Array kann ja nur von einem Typ sein). Die Komponenten eines Arrays werden angegeben durch den entsprechenden Variablennamen und einem Ausdruck in eckigen Klammern, der den Wert des Indexes bestimmt. Um diesen Sachverhalt genau zu verstehen, sei hier ein Beispiel angegeben, das ein Array von 50 Elementen definiert:

```
TYPE bereich = [1..50];
        werte = ARRAY bereich OF integer;
 VAR element: werte;
```

In der »TYPE«-Anweisung haben wir den Bereich in einem eigenen Typ »bereich« festgelegt. Dieser wird nun als Indextyp für eine Array-Typdefinition mit dem Namen »werte« verwendet. In der Variablendeklaration fungiert der Array-Typ »werte« schließlich als Datentyp für die Variable »element«, die somit zu einer Array-Variablen mit 50 Komponenten wird. Es ist jedoch nicht nötig, eine einfache Array-Definition wie diese in solch vielen Schritten zu tun. Pascal erlaubt das Schlüsselwort »ARRAY« auch direkt in der Variablendeklaration, so daß man auch schreiben kann:

```
VAR element: ARRAY [1..50] OF integer;
```

Diese sehr viel kürzere Angabe hat die gleiche Wirkung. Sie ist im Rahmen dieses Beispiels auch noch recht übersichtlich. Bei komplexeren Datentypen (zum Beispiel zusammengesetzten Typen) ist eine Aufschlüsselung wie oben im Sinne der Überblickbarkeit angebrachter. Nachdem wir die Variable »element« als ein Array definiert haben, können einzelne Komponenten mit ihrem Index angegeben werden. So ist

```
element[10]
```

das zehnte Element des Feldes. Innerhalb der eckigen Klammern darf aber auch ein Ausdruck stehen und so kann das Element Nummer 10 auch folgendermaßen dargestellt werden:

```
element[2 * 5]
```

Es ist sogar möglich, andere Variablen (zum Beispiel »x«) im Indexausdruck zu benutzen, sofern diese vom geeigneten Typ sind (in diesem Fall integer):

```
element[3 + 2 * x]
```

Jede dieser Komponenten ist eine Variable des Grundtyps des Arrays (in unserem Beispiel des Typs integer) und kann als solche wie jede andere Integer-Variable verarbeitet werden.

```
element[5] := element [x * 2];
element[3 * (x + 1)] := element[x] - element[x - 1];
```

Um die Arbeit mit Arrays zu verdeutlichen, ist in Listing 12 ein Beispielprogramm abgedruckt, das die Eingabe von 50 Integer-Zahlen in ein Array ermöglicht und schließlich die Summe der Werte ausgibt. Zusammen mit dem Index bestimmt der Name des Arrays (in unserem Beispiel »element«) die einzelnen Komponenten des Feldes. Der Arrayname für sich allein bezeichnet dagegen das gesamte Feld. Auf ihn selbst lassen sich keine Operatoren anwenden, mit Ausnahme des Zuweisungsoperators »:=«. Mit ihm kann ein

```
PROGRAM eingabe;
VAR daten: ARRAY  1..50  OF integer;
    index: integer;
    BEGIN
      FOR index := 1 TO 50 DO BEGIN
          WRITELN(index,',');
          READLN(daten index );
      END;
    END.
```

Listing 12. Arrays – Felder

ganzes Feld einem anderen Feld des gleichen Datentyps zugewiesen werden:

```
TYPE werte = ARRAY [1..50] OF integer;
VAR i,k: werte;
```

Mit den obigen Vereinbarungen kann man den Zuweisungsoperator auf die Array-Variablen »i« und »k« anwenden:

```
i := k;
```

Damit erhalten alle Elemente des Arrays »i« die Werte der entsprechenden Elemente des Arrays »k«. Wie alle anderen Variablen haben auch Array-Variablen nach ihrer Definition noch keine Werte. Sie müssen, wie jede einfache Variable auch, zuerst einen Wert zugewiesen bekommen. In vielen Fällen erhalten die Elemente eines Feldes zu Beginn eines Programmablaufs die gleichen Werte. Während das bei den einfachen Variablen durch einzelne Zuweisungen geschehen muß, ist bei Arrays die Verwendung der FOR-Schleife recht praktisch:

```
FOR x := 1 TO 50 DO k[x] := 100;
```

Möchte man, daß das Array »i« dieselben Anfangswerte annimmt, genügt danach die bereits erwähnte einfache Zuweisung

```
i := k;
```

Wie schon einmal erwähnt, ist es möglich, daß Arrays wiederum Komponententypen von weiteren Arrays sein können. So ist dies im Rahmen des Speicherplatzes in Pascal beliebig oft durchführbar. Man kann zum Beispiel folgendes Array definieren:

```
TYPE index = [1..25];
     feld1 = ARRAY [index] OF integer;
     feld2 = ARRAY [index] OF feld1;
VAR a,b: feld2;
```

Mit diesen Eingaben werden die Variablen »a« und »b« als Arrays des Typs »feld2« definiert. »feld2« ist aber gleichzeitig ein Arraytyp des Typs »feld1«, das wiederum ein Array ist. Dieses Spiel könnte man endlos fortsetzen, bis man schließlich den Überblick verliert.

```
ARRAY [Index1] OF ARRAY [Index2] OF ARRAY
[Index3] OF ARRAY [Index4] OF ...
```

Um eine gewisse Übersicht zu wahren, können solch verschachtelte Arrays auch in einer Kurzschreibweise angegeben werden, indem die Indizes in den eckigen Klammern stehend hinter das Wort »ARRAY« geschrieben werden. Das folgende Schema verdeutlicht dies:

```
ARRAY [Index1,Index2,Index3,Index4,...]
OF (Komponententyp)
```

Das Ansprechen der einzelnen Elemente eines solchen Arrays wird ähnlich durchgeführt. Man schreibt hier:

```
element[a1,a2,a3,a4,...]
```

Da diese Felder mehr als einen Index zur Unterscheidung haben, werden sie auch »mehrdimensionale Arrays« genannt. Sie werden sich nun fragen, was man mit solchen Feldern anfangen kann. Ein Beispiel soll dies verdeutlichen: Das Problem ist eine Tabelle, die den Preis für ein Produkt von 1978 bis 1980 für jeden der 12 Monate enthalten soll. Zum Schluß soll der Durchschnittspreis für 1979 ermittelt werden. Hierfür benötigt man ein zweidimensionales Array, das heißt ein Feld von Werten, die durch zwei Indizes unterschieden werden. Listing 13 zeigt die Lösung dieser Aufgabe.

Im allgemeinen kann man sagen, daß sich Probleme, die viele Werte festhalten müssen, welche wiederum von verschiedenen Faktoren abhängen, am elegantesten mit mehrdimensionalen Arrays lösen lassen.

```
PROGRAM KOSTEN;
  VAR PREIS:ARRAY[1978..1980,1..12] OF INTEGER;
    MONAT,JAHR,GESAMT:INTEGER;
BEGIN
  WRITELN('GEBEN SIE DIE TABELLE EIN');
  FOR JAHR:=1978 TO 1980 DO
    FOR MONAT:=1 TO 12 DO
      READ(PREIS[JAHR,MONAT]);
  (* BERECHNE DEN DURCHSCHNITTSPREIS FUER 1979 *)
  READLN;
  GESAMT:=0;
  FOR MONAT:=1 TO 12 DO
    GESAMT:=GESAMT+PREIS[1979,MONAT];
  WRITELN('DURCHSCHNITTSPREIS 1979:',ROUND(GESAMT/12))
  (* ZUSAETZLICHE ANWEISUNGEN FUER DIE BERECHNUNG *)
  (* ANDERER DURCHSCHNITTSWERTE HIER ANFUEGEN *)
END.
```

Listing 13. Kostenberechnung

Felder und gepackte Daten

Große Felder benötigen, wie auch andere strukturierte Datentypen (wir werden noch einige kennenlernen), sehr viel Speicherplatz. In Pascal ist es möglich, eine knappere Speicherungsart als die übliche zu erreichen. Dadurch verliert der Zugriff auf diese Daten einiges an Effizienz. Wenn es jedoch darauf ankommt, Speicherplatz zu sparen, muß man die Daten zusammen-»packen«. In Pascal steht hierfür das Schlüsselwort »PACKED«, das dem strukturierten Typ vorangestellt wird. Für gepackte Arrays gilt demnach folgende Syntax:

```
PACKED ARRAY [Indextyp] OF (Komponententyp)
```

Ein in Standard-Pascal wichtiger gepackter Typ ist:

```
PACKED ARRAY [1..n] OF char
```

Auf diese Weise wird nämlich der Datentyp string der Länge eins bis n definiert. Für UCSD-Pascal-Programmierer ist dieser Ausdruck ziemlich belanglos, da der UCSD-Standard den Datentyp string bereits vordefiniert hat und eine sehr viel komfortablere Handhabung erlaubt. Darum soll im Rahmen dieses Kurses nicht weiter darauf eingegangen werden. Die Elemente von gepackten Feldern haben die gleichen Bezeichnungen wie die von ungepackten und werden auch so gehandhabt. Anweisungen, die gepackte Daten benützen, können jedoch sehr lange dauern. Der Vorteil dabei ist aber die Speicherplatzersparnis. Das Wort »PACKED« bezieht sich immer nur unmittelbar auf das nachfolgende Wort. So muß man bei Schachtelungen besondere Vorsicht walten lassen:

```
PACKED ARRAY [Index1] OF ARRAY [Index2]
OF (Komponententyp)
```

ist nur auf der ersten Schachtelungsstufe (für Index1) gepackt. Soll dies in der zweiten Stufe auch geschehen, so muß man ein zweites »PACKED« hinzufügen:

```
PACKED ARRAY [Index1] OF PACKED ARRAY [Index2]
OF (Komponententyp)
```

Die schon bekannte Kurzschreibweise ist wesentlich einfacher und übersichtlicher:

```
PACKED ARRAY [Index1,Index2] OF (Komponententyp)
```

Es ist in Pascal möglich, ein bereits als ungepackt oder gepackt definiertes Feld nachträglich zu packen oder zu entpacken. Pascal bietet dazu zwei Datentransferprozeduren, die diese Arbeit übernehmen. Das Packen eines normalen Arrays erfolgt mit der Prozedur

```
PACK (up,ix,pa)
```

Diese Prozedur überträgt den Komponenten des gepackten Feldes »pa« die Werte der Komponenten des ungepackten Feldes »up« von Index »ix« bis zum Ende. Es muß darauf geachtet werden, daß beide Felder vom gleichen Komponententyp sind. Ebenso darf das gepackte Feld höchstens so viele Elemente haben wie das ungepackte, auf das die Daten übertragen werden. Die Umkehrung dazu geht mit der Prozedur

```
UNPACK (pa,up,ix)
```

vonstatten. Es ist aber zu beachten, daß hier zuerst das gepackte Feld »pa«, dann das ungepackte Feld »up« und

schließlich der Index »ix« angegeben werden muß. Will man einzelne Werte eines Arrays ver- oder entpacken, genügt der Zuweisungsoperator. Es genügt

```
up(Index) := pa(Index);
```

zum Entpacken und

```
pa(Index) := up(Index);
```

zum Packen, sofern beide Felder vom gleichen Komponententyp sind. Die Zuweisung von ganzen gepackten Feldern auf ungepackte Felder ist mit dem Zuweisungsoperator jedoch nicht möglich. Der Ausdruck

```
up := pa;
```

ist zum Beispiel unzulässig. Für diese Zwecke müssen die Prozeduren »PACK« und »UNPACK« verwendet werden. Man sollte Arbeiten mit gepackten Feldern möglichst unterlassen, da sie sehr zeitaufwendig werden können. Bevor auf gepackte Daten zugegriffen wird, ist es ratsam, diese vorher mit »UNPACK« zu entpacken und nach den Operationen wieder zu packen, um die Effizienz möglichst hoch zu halten. Das Packen und Entpacken ist, wie bereits angeschnitten, auch mit sämtlichen anderen strukturierten Datentypen möglich. Bisher kennen wir ja nur Arrays, doch werden Sie gegen Ende des Kurses noch mit weiteren Datentypen konfrontiert werden. Doch »packen« wir zunächst ein weiteres wichtiges Merkmal von Pascal an, die Blockstruktur.

Pascal und Unterprogramme

Es gibt in Pascal auch eine Art Unterprogrammtechnik, die es erlaubt, an verschiedenen Stellen mehrmals einen gewissen Programmabschnitt ablaufen zu lassen. Die Unterprogramme bekommen in Pascal Namen, mit denen sie gemäß ihrer Aufgabe aufgerufen werden können. Diese Art des Programmierens erlaubt eine Unterteilung des Programms in einzelne unabhängige Blöcke, von denen jeder ein gewisses Teilproblem des Gesamtprogramms löst. Man bezeichnet dies auch als »Blockstruktur«, was ein Hauptkriterium des strukturierten Programmierens ist. Es gibt in Pascal zwei verschiedene Arten von Unterprogrammen, die Funktionen und die Prozeduren. Ihre Bedeutung soll nun erläutert werden.

Prozeduren sind bei größeren Programmen besonders häufig. Sie machen es möglich, unabhängige Unterprogramme mit einem Namen zu versehen, so daß sie wie eine neue Pascal-Anweisung durch Aufruf des Namens im Hauptprogramm benutzt werden können. Einige Standardprozeduren, die in UCSD-Pascal bereits vordefiniert sind, haben Sie ja schon kennengelernt. In Bild 3 sind sie unter »Standardprozeduren« aufgelistet. Bevor eine Prozedur jedoch aufgerufen werden kann, muß sie selbstverständlich noch geschrieben werden. Dies geschieht vor Beginn des Hauptprogramms mit dem Schlüsselwort »PROCEDURE«. Allgemein hat eine Prozedur-Deklaration die folgende Form:

```
(Prozedur-Kopf); (Anweisungsblock)
```

Der Prozedur-Kopf wird hierbei mit dem Wort »PROCEDURE« eingeleitet:

```
PROCEDURE (Name) ( (formale Parameter) );
```

Hinter dem Schlüsselwort »PROCEDURE« muß der Name der Prozedur stehen. Er dient später zur Identifikation. Von runden Klammern umgeben, kann schließlich eine Liste von formalen Parametern folgen. Diese gliedern sich in zwei Arten, die Eingabe- und Ausgabeparameter. Sie müssen aber, wenn sie nicht gebraucht werden, auch nicht angegeben werden. Danach folgt der Anweisungsblock, der wie eine Verbundanweisung mit »BEGIN« angefangen und mit »END;« abgeschlossen werden muß. Er besteht aus beliebigen Anweisungen, die während des Prozedurablaufs abgearbeitet werden sollen. Die Angabe des Semikolons hinter »END« ist dabei sehr wichtig. Der Prozeduraufruf erfolgt durch die Nennung des Prozedurnamens und den in runden Klammern stehenden aktuellen Parametern, die die Prozedur benötigt. Bei Aufruf der Prozedur werden die Parameter, wenn vorhanden, übergeben und, wenn nötig, ein oder mehrere Werte an das Hauptprogramm zurückgeliefert. Um die Bedeutung der Ein- und Ausgabeparameter zu erklären, sei hier eine kleine Prozedur als Beispiel angegeben, die die größere von zwei Integerzahlen ermittelt und dem Hauptprogramm zur Verfügung stellt:

```
PROCEDURE groesser (zahl1,zahl2: integer;
                    VAR ergebnis: integer);
BEGIN
  IF zahl1 > zahl2 THEN ergebnis := zahl1
                   ELSE ergebnis := zahl2
END;
```

Die Prozedur wird in unserem Beispiel »groesser« genannt. Die formalen Eingabeparameter sind die Variablen »zahl1« und »zahl2«, der Ausgabeparameter ist »ergebnis«. Er wird auch als »VAR«-Parameter bezeichnet, da er mit dem Schlüsselwort »VAR« kenntlich gemacht werden muß. Diese drei Parameter stellen nun die einzigen Verbindungen zum restlichen Programm dar. Ausgabeparameter können dazu dienen, Daten in die Prozedur einzulesen und auch auszugeben. Eingabeparameter sind jedoch nur zur Eingabe von Daten gedacht. Die Namen der Parameter »zahl1«, »zahl2« und »ergebnis« gelten dabei nur im Zusammenhang mit ihrer Prozedur und müssen deshalb nicht im Deklarationsteil definiert werden. Man nennt sie auch lokale Variablen, da sie nur für diese eine Prozedur lokal verfügbar sind. Die Variablen, die im normalen Deklarationsteil zu Beginn des Gesamtprogramms deklariert wurden, sind dagegen im Hauptprogramm wie in den Unterprogrammen, den Prozeduren und Funktionen (die wir noch behandeln werden), immer verwendbar. Sie heißen aus diesem Grund auch »globale Variable«, da sie global für alle Programmteile zur Verfügung stehen. Konstante sind im Gegensatz dazu immer global, das heißt überall benutzbar. Wir können nun unsere Prozedur im Hauptprogramm aufrufen, indem wir den Namen »groesser«, gefolgt von drei aktuellen Parametern, angeben. Die aktuellen Parameter sind dabei völlig andere als die formalen in der Deklaration. Sie sind normal definierte Variablen, die jeweils aktuelle Werte enthalten, mit denen die Prozedur ausgeführt werden soll. Wir können zum Beispiel unter der Voraussetzung, daß die verwendeten Variablen korrekt deklariert wurden, schreiben:

```
groesser(ersterwert,zweiterwert,groessterwert);
```

Beim Aufruf der Prozedur werden die Werte der aktuellen Parameter in den runden Klammern an die formalen Parameter, das heißt also an die lokalen Variablen der Prozedur, übergeben. Aktuelle und formale Parameter können deshalb vollkommen verschiedene Namen haben. Dies ist ja auch in unserem Beispiel der Fall. Die Zuordnung der aktuellen zu den formalen Parametern richtet sich nur nach der Reihenfolge des Aufschreibens. In unserem Beispiel erfolgt die Zuweisung folgendermaßen:

zahl1 erhält den Wert von ersterwert
zahl2 erhält den Wert von zweiterwert
ergebnis erhält den Wert von groessterwert

Die Anzahl und jeweils deren Typ müssen dabei selbstverständlich mit denen der formalen Parameter in der Prozedur-Deklaration übereinstimmen. In unserem Fall müssen alle Variablen vom Typ integer sein. In der Prozedur wird der lokalen Variable »ergebnis« der größte Wert der beiden Variablen »zahl1« und »zahl2« zugewiesen. Bei Verlassen der Prozedur übergibt die Variable »ergebnis« ihren Inhalt schließlich an die Variable »groessterwert«, die nun im Hauptprogramm weiterverarbeitet werden kann. »ergebnis« ist ja der formale Ausgabeparameter und »groessterwert« die dazugehörige aktuelle Variable, die den Augabewert erhält. Listing 14 zeigt die Anwendung unserer Beispielprozedur auf recht einfache Weise.

Werden in einer Prozedur weitere Variablen als die Ein- und Ausgabeparameter benötigt, so können diese direkt hinter dem Prozedur-Kopf durch das schon bekannte Schlüsselwort »VAR« definiert werden. Die in einer Prozedur extra deklarierten Variablen sind wie die formalen Parameter nur dieser einen Prozedur zugänglich, also ebenfalls lokal. Eine Prozedur mit einer lokalen Variablendefinition wäre zum Beispiel folgende:

```
PROCEDURE sternchen (laenge: integer);
 VAR index: integer;
 BEGIN
   FOR index := 1 TO laenge DO
        WRITE('*');
 END;
```

Beim Aufruf der Prozedur »sternchen« wird eine Reihe von Sternen (»*«) ausgegeben, deren Anzahl vom Hauptprogramm an den Eingabeparameter »laenge« übergeben wird. Die Anweisung

```
sternchen(20);
```

im Hauptprogramm bewirkt dann zum Beispiel den Ausdruck von 20 Sternchen auf dem Bildschirm. Die lokal definierte Variable »index«, die zur »FOR«-Schleife benötigt wird, ist nur für die Prozedur sternchen zugänglich und wird nach Abschluß der Prozedur wieder vollkommen aus dem Rechenspeicher gelöscht. Erst bei Wiederaufruf von »sternchen« wird diese Variable erneut angelegt.

Es ist möglich, innerhalb einer Prozedur-Deklaration weitere Prozeduren oder Funktionen zu deklarieren. Auch sie sind dann lokal und können nur innerhalb dieser Prozedur aufgerufen werden. Hierzu unsere schon bekannte Prozedur mit einer kleinen Erweiterung:

```
PROCEDURE sternchen (laenge: integer);
  VAR index: integer;
(* Definition der lokalen Prozedur anzahl *)
  PROCEDURE anzahl (sterne: integer);
   BEGIN
   WRITE ('Das sind genau ',sterne,'Sternchen.');
   END;  (* Ende der Prozedur anzahl *)
(* Beginn des Anweisungsteils von sternchen *)
BEGIN
  FOR index := 1 TO laenge DO BEGIN WRITE('*');
      WRITELN;
      anzahl(laenge);
  END;
END;
(* Ende von sternchen *)
```

Die Prozedur »sternchen« hat hier eine lokale Prozedur-Deklaration mit dem Namen »anzahl« bekommen, die die genaue Anzahl der Sterne in der ausgedruckten Reihe ausgibt. Der Aufruf von »anzahl« ist dabei nur innerhalb der Prozedur »sternchen« selbst erlaubt. Wollte man »anzahl« vom Hauptprogramm aus benutzen, so würde dies zu einer Fehlermeldung des Computers führen, da er die lokale Prozedur »anzahl« außerhalb von »sternchen« wieder »vergessen« hat. Eine Prozedur kann also als in sich abgeschlossenes unabhängiges Teilprogramm angesehen werden, sofern sie nicht globale Variablen oder Konstanten benutzt, die vom Hauptprogramm aus deklariert worden sind. Die einzige Verbindung nach außen besteht dabei in den globalen Variablen und den formalen Parametern zur Übergabe von Werten. Die zweite Art von Unterprogrammen ist den Prozeduren sehr ähnlich.

```
PROGRAM maximum;
VAR erstezahl, zweitezahl, groesste: integer;
PROCEDURE groesser (zahl1,zahl2: integer,
                    VAR max: integer);
 BEGIN
  IF zahl1 ( zahl2 THEN max := zahl2
                   ELSE max := zahl1;
 END;
 (* Anfang Hauptprogramm *)
BEGIN
   READ(erstezahl,zweitezahl);
   groesser(erstezahl,zweitezahl,groesste);
   WRITE('Die groessere von beiden ist ', groesste);
END.
 (* Ende Hauptprogramm *)
```

Listing 14. Prozeduren

Funktionen werden den meisten aus dem Mathematikunterricht bekannt sein. So haben sich bestimmt einige schon mit den trigonometrischen Funktionen Sinus, Cosinus und Tangens herumgeschlagen oder die Exponential- und Logarithmusfunktionen hassen gelernt. In Pascal sind Funktionen jedoch ein wichtiger Begriff. Zu Beginn des Kurses wurden ihnen schon einige standardmäßig vordefinierte Funktionen vorgestellt. Sie sind auch in Bild 3 unter »Standardfunktionen« aufgelistet. Jetzt soll geklärt werden, wie diese genau zu handhaben sind und wie man eigene Funktionen schreiben kann. Im Gegensatz zu Prozeduren liefern Funktionen automatisch einen Wert an das Hauptprogramm zurück. Sie haben aber einen ähnlichen Aufbau wie Prozeduren. Eine Funktionsdeklaration erfolgt nach den Konstanten-, TYPE- und Variabeldefinitionen und wird mit dem Schlüsselwort »FUNCTION« eingeleitet. Sie bekommt in Pascal ebenfalls einen Namen, mit dem die Funktion später vom Hauptprogramm oder anderen Unterprogrammen aufgerufen werden kann. Die allgemeine Form einer Funktion kann folgendermaßen beschrieben werden:

(Funktionskopf); (Anweisungsteil)

Der Funktionskopf sieht dabei so aus:

```
FUNCTION (Name) ( (Liste von formalen
Parametern) ): (Typ des Resultats)
```

Hier zeigt sich schon der erste Unterschied zur Prozedur. Es muß zusätzlich der Typ des Resultats der Funktion angegeben werden. Die Liste der formalen Parameter kann wie bei »PROCEDURE« fehlen, wenn es sich um eine Funktion handelt, die keine Argumente braucht, um einen Wert zu liefern. Man nennt eine solche Funktion auch »nullstellige Funktion«. Dies ist jedoch sehr selten. Eine Anwendung wäre zum Beispiel eine Time-Funktion, die bei Aufruf immer die aktuelle Zeit angibt. Normalerweise wird aber immer eine Liste an formalen Parametern benötigt werden. Um den Unterschied einer Funktion zu einer Prozedur zu erläutern, soll hier unser bekanntes Beispiel (ein Programmteil, das die größere von zwei Integer-Zahlen ermittelt) als Funktion geschrieben werden:

```
FUNCTION groesser (zahl1,zahl2: integer): integer;
  BEGIN
    IF zahl1 > zahl2 THEN groesser := zahl1
                     ELSE groesser := zahl2
  END;
```

Hier tut sich nun etwas sehr seltsames. Wie die Prozedur, so wird auch die Funktion »groesser« mit einer Liste von formalen Eingabeparametern definiert. Unklar ist im Moment aber die zusätzliche Angabe des Resultattyps (integer) hinter der eingeklammerten Liste. Wenn wir die Funktion weiter verfolgen, wird sich dieses Rätsel jedoch bald lösen. Wie in der uns bekannten Prozedur wird eine »IF...THEN...ELSE«-Verzweigung zur Ermittlung der größeren Zahl verwendet. Doch statt die nun größte Zahl in eine Variable (bei der Prozedur die Variable »ergebnis«) zu schreiben, wird die größte Zahl dem Funktionsnamen selbst zugewiesen. Die Funktion wird also wie eine Variable behandelt. Jetzt wird auch die Angabe des Resultattyps klar. Er bestimmt, von welchem Typ der Wert der Funktion sein darf. Im Hauptprogramm wird bei einem Aufruf die Funktion wie ein Wert verstanden und in die Operationen und Variablenzuweisungen direkt eingebunden. Rechnerintern wird die Funktion selbst jedoch nicht als Platz-

halter für einen Wert verwendet, sondern eine lokale Variable mit dem selben Namen wie die Funktion eingerichtet. Eine Funktion kann zum Beispiel folgendermaßen aufgerufen werden:

```
maximum := groesser(erstezahl,zweitezahl);
```

Die Variable »maximum« erhält damit das Resultat der Funktion »groesser« zugewiesen. Der Resultattyp und der Typ der Variablen »maximum« muß dabei logischerweise der gleiche sein. In Listing 15 ist das Programm »groesser« dieses Mal mit einer Funktion formuliert worden. Es hat die gleiche Wirkung wie das Programm in Listing 14, zeigt aber die markanten Unterschiede von Funktionen und Prozeduren.

Damit ist die Verwendung der Standardfunktionen ebenfalls geklärt, da auch sie nach dem gleichen Prinzip aufgerufen werden, wie zum Beispiel:

```
y := sin(x);
```

oder

```
v := succ(v);
```

Die Regeln von lokalen und globalen Größen sind die gleichen wie bei Prozeduren. Auch hier sind die formalen Parameter und in der Funktions-Deklaration definierte Variablen lokal. Es ist ebenfalls möglich, innerhalb von Funktionen weitere lokale Funktionen oder Prozeduren zu definieren, die nur dieser Funktion zugänglich sind. Der kombinierten Anwendung von Funktionen und Prozeduren sind somit keine Grenzen gesetzt.

Es gibt in der Mathematik manchmal Probleme, die auf Anhieb nicht so ohne weiteres in Programmen gelöst werden können. Möchte man zum Beispiel eine Funktion in Pascal definieren, die die Fakultät einer Integer-Zahl berechnet, so kann dies einige Schwierigkeiten ergeben, wenn man nur folgende Definition der Fakultät zur Verfügung hat:

$n! = 1$ für den Fall $n = 0$, und

$n! = n \cdot (n-1)!$ Für den Fall $n > 0$.

Man sieht an dieser Definition, daß zur Berechnung von n! für den Fall $n > 0$ wiederum eine Fakultätsberechnung nötig ist. Es stellt sich jetzt die Frage, ob man dies direkt in ein Pascal-Programm umsetzen kann. Die Funktion würde dann wie folgt aussehen:

```
FUNCTION fak (n: integer): integer;
  BEGIN
    IF n = 0 THEN fak := 1
             ELSE fak := n * fak(n - 1)
  END;
```

Man erkennt, daß innerhalb der Funktion »fak« dieselbe nochmals aufgerufen wird (fak(n - 1)). So seltsam diese Lösung auch scheinen mag, sie ist in Pascal möglich. Man nennt diese Funktion, da sie sich selbst aufruft, auch »rekursive Funktion« (Rekursion: auf sich zurückgreifend). Man fragt sich hierbei natürlich, ob man damit wirklich eine Lösung erhält. Um dies zu überprüfen, gehen wir das Programm, welches diese rekursive Funktion benutzt, Schritt für Schritt durch. Listing 16 zeigt das Programm zur rekursiven Berechnung der Fakultät.

Nehmen wir der Einfachheit halber die Berechnung der Fakultät von 3. Das Hauptprogramm benötigt zur Berechnung drei Integer-Variablen mit den Namen »argument«, »ergebnis« und »fak«, die das Ergebnis der Funktion »fak« beinhalten soll. Wenn wir in das Programm den Wert 3 eingeben, so wird »argument« damit belegt. Dies ist im Schema von Bild 9 dargestellt.

Wird die Funktion »fak(argument)« aufgerufen, wird ein zweiter Block eröffnet, der eine neue Variable mit dem Namen »n« benötigt. Er übernimmt den Wert von »argument«. Fügen wir dies also unserem Schema hinzu (Bild 10).

Die Variable »n« ist nicht Null. Aus diesem Grund wird die Anweisung hinter »ELSE« ausgeführt. Diese ruft aber wiederum die Funktion »fak()« auf:

```
fak := n * fak(n - 1);
```

Die Zeile mit dem Wert »fak« stellt in unserem Schema den Wert dar, der sich bei der obigen Berechnung ergibt. Doch bevor sich ein Ergebnis einstellt, wird fak (n – 1) berechnet. Man muß jetzt annehmen, daß innerhalb des Blocks »fak« ein zweiter eröffnet wird, der die Bearbeitung von »fak(n – 1)«,

```
PROGRAM maximum;
VAR erstezahl,zweitezahl,groesste: integer;
 FUNCTION groesser (zahl1,zahl2: integer): integer;
   BEGIN
     IF zahl1 <zahl2 THEN groesser := zahl2
                     ELSE groesser := zahl;
   END;
    (* Anfang Hauptprogramm *)
    BEGIN
      READ(erstezahl,zweitezahl);
      groesste := groesser(erstezahl,zweitezahl);
      WRITE('Die groessere von beiden ist ',groesste);
    END.
    (* Ende Hauptprogramm *)
```

Listing 15. Funktionen

```
PROGRAM fakultät;
VAR argument,ergebnis:integer;
  FUNCTION fak (n: integer): integer;
    BEGIN
      IF n = 0 THEN fak := 1
               ELSE fak := n * fak(n - 1);
    END;
(* Anfang Hauptprogramm *)
BEGIN
     READ(argument);
     ergebnis := fak(argument);
END.
       (* Ende Hauptprogramm *)
```

Listing 16. Fakultät rekursiv

argument: 3
ergebnis:
fak:
ergebnis:= fak (argument)

Bild 9. Struktogramm zum Programm Fakultät

argument: 3
ergebnis:
fak:
fak n: 3 IF n = 0 THEN fak: =1 ELSE fak: =n · fak (n–1)
ergebnis:= fak

Bild 10. Struktogramm zur Fakultät mit Funktion

also »fak(2)«, übernimmt. Der Computer errichtet dazu eine zweite Variable des Namens »n«, die aber nur lokal für den zweiten Funktionsaufruf gilt. Es muß ebenfalls eine zweite Funktionsvariable »fak« geben, die das Ergebnis des zweiten Durchgangs aufnimmt. Um klarzustellen, daß es sich bei den gleichnamigen Variablen der verschiedenen Rekursionsschichten um unterschiedliche Speicherplätze handelt, werden in den Bildern Indizes verwendet. Bild 11 hat dies im Schema festgehalten.

Wiederum ist »n« nicht Null, und somit wird die Anweisung hinter »ELSE« ausgeführt. Doch bevor »fak1« berechnet werden kann, muß der Computer noch eine Rekursionsstufe tiefer steigen, wo ihn die gleiche Situation für »n« gleich 1 erwartet. Nach dem dritten rekursiven Aufruf ist das lokale »n« gleich Null, womit »fak3« den eindeutigen Wert 1 erhält. Die Berechnung der Rekursionsformel in den verschiedenen Tiefen können jetzt gelöst werden, und der Computer kann wieder auf die ursprüngliche Ebene zurückkehren, wo er der Variablen »ergebnis« den Wert von »fak« zuweisen kann. »ergebnis« erhält den Wert 6, was der Fakultät von 3 entspricht. Bild 12 verdeutlicht alle nötigen Rekursionsebenen des Problems.

Schon allein das gedankliche Nachvollziehen einer solchen Berechnung zeigt, daß es einen unwahrscheinlich großen Aufwand für den Computer bedeuten muß, eine rekursive Aufgabe zu bearbeiten. Da dies noch ein relativ einfaches Problem ist, kann man sich vorstellen, daß komplexere Rekursionsaufgaben die Rechenzeit erheblich erhöhen können. Wem es also auf Effizienz im Programm ankommt, der sollte Rekursionen möglichst vermeiden. Diese Art der Lösung eines Problems ist jedoch sehr elegant und für den Anwender oft recht einfach zu programmieren. Über die komplizierten Verwaltungsaufgaben des Computers muß sich der Anwender ja keine Gedanken machen. Einige Probleme lassen sich oft nur durch Rekursion bewältigen, und hierfür ist Pascal eine gute Programmiersprache. Die Rekursion läßt sich auch auf Prozeduren anwenden. Die einzelnen Schritte dafür sind sehr ähnlich. Darum wird an dieser Stelle nicht mehr gesondert darauf eingegangen.

Wie bereits schon angeschnitten wurde, gibt es außer dem Datentyp »array« noch einige weitere strukturierte Typen, die jetzt erklärt werden sollen. Doch zuvor geben wir einen kleinen Überblick über alle verwendbaren Datentypen. In Pascal gibt es für Datentypen drei Überbegriffe: Da sind zunächst die einfachen Typen, von denen wir schon alle erläutert haben. Sie sind in Bild 12 unter dem Begriff »einfach« aufgelistet. Die zweite Art von Typen nennt sich »strukturiert«. In Bild 12 wurden sie unter »strukturiert« zusammengefaßt. Bis auf den Array-Typ dürften Ihnen diese Typen noch unbekannt sein; sie werden Gegenstand dieses Abschnittes. Als letztes stehen noch die sogenannten »Pointer« oder »Zeiger« zur Verfügung, denen der nächste Abschnitt gewidmet ist. Kommen wir zunächst zu den »structured types«, den strukturierten Datentypen. Wie bereits bei Arrays erklärt wurde, lassen sich alle strukturierten Typen mit dem Schlüsselwort »PACKED« in ihrer Speicherung komprimieren. Dies geschieht bei den noch unbekannten Typen genauso, wie wir das von Arrays her kennen. Das gleiche gilt auch für das nachträgliche Packen und Entpacken. Aus diesem Grund wird bei der Besprechung der weiteren Typen nicht mehr extra darauf eingegangen. Die folgenden Angaben beziehen sich also auf ungepackte Typen.

Der Set-Typ

Der Set-Typ ermöglicht eine Verarbeitung von mehreren Komponenten des gleichen Typs, die zu einer Menge zusammengefaßt werden. Man definiert einen Set-Typ mit den Schlüsselworten »SET OF« in der folgenden Weise:

```
SET OF (Basistyp)
```

Der Basistyp darf hierbei jeder einfache Typ sein, mit Ausnahme von real. Eine Variable dieses Typs kann jede Menge von Werten des Basistyps annehmen. So kann sie auch eine Teilmenge, also mehrere Elemente der mit »SET« definierten Grundmenge, enthalten. Die maximale Anzahl von Werten einer Menge ist bei den meisten Pascal-Implementationen begrenzt und von Compiler zu Compiler verschieden. Eine korrekte Definition wäre zum Beispiel

```
TYPE menge = SET OF char;
```

Ein so definierter Mengentyp kann nur im Gesamten angesprochen werden. Es ist demnach nicht möglich, einzelne Elemente der Menge zu verabeiten, wie man dies zum Beispiel bei Arrays kann. Eine Menge wird in Pascal durch eine Aufzählung von Elementen oder Elementbereichen, die in eckigen Klammern durch Kommata getrennt sind, beschrieben. Mengen des Typs char sind beispielsweise:

```
['a'..'z']   ist die Menge aller Buchstaben
['a','c','e'] ist die Menge der Zeichen 'a', 'c'
und 'e'
[] ist die leere Menge
```

Die Definition des Typs set wird nach dem in Pascal üblichen Schema vorgenommen:

```
TYPE farben = [rot,gelb,gruen,blau,braun,lila,
rosa,grau,tuerkis,karmin];
menge = SET OF farben;
VAR schoen,haesslich: menge;
```

Damit wurde den Variablen »schoen« und »haesslich« der Typ »menge« zugeordnet, der ein »SET« des Basistyps »farben« ist. Die Variablen können nun Mengen der Werte des Typs »farben« enthalten. Die Zuweisung von Mengen

```
argument: 3
ergebnis:
fak:
fak
   n: 3
   fak1:
   fak
      n: 2
      IF   n = 0 THEN
           fak1: = 1
                ELSE
           fak1: =
           n · fak (n–1)
   fak:= n · fak1
ergebnis:= fak
```

Bild 11. Rekursionsschichten

Datentyp		
einfach	strukturiert	Zeiger
integer real boolean char string	ARRAY SET RECORD FILE	

Bild 12. Alle Datentypen auf einen Blick

geschieht auch hier wie immer durch den Zuweisungsoperator (»:=«).

```
schoen := [rot..blau,rosa,tuerkis,karmin];
haesslich := [braun,lila,grau,gelb];
```

Solche Mengen lassen sich nun auch manipulieren. Es stehen dafür einige Operatoren zur Verfügung, die in bezug auf die Zeichen bereits bekannt sind, bei den Mengentypen jedoch eine andere Bedeutung haben. Bild 13 zeigt diese Operatoren und ihre Bedeutung.

Diese Operatoren können auf die in Pascal übliche Weise angewandt werden und Ausdrücke bilden, wie zum Beispiel:

```
[rot..blau] * [gruen]
schoen + [grau]
schoen <= [gelb]
(tuerkis) IN haesslich
```

Konstanten des Mengentyps können auch ohne vorherige Vereinbarung mit »SET« in Pascal-Programmen verwendet werden. So sind die Mengen

```
[1..100]  [5..10]  ['c'..'v']
```

ohne vorherige Definitionen zulässig. Mengen, wie

```
[3.7,4.7,5.7]  [100 / 5..50]
```

sind dageben verboten, da keine reellen Zahlen für Mengenangaben gestattet sind. Auch eine gemischte Menge darf nicht vorkommen, da alle Elemente einer Menge vom gleichen Typ sein müssen.

```
['a'..'z',2..10]
```

ist somit unzulässig. Mengen bieten die Möglichkeit, einige Probleme leichter und durchsichtiger zu gestalten. Um festzustellen, ob ein Zeichen zum Beispiel eine Ziffer ist, mußte man mit den herkömmlichen Methoden folgende Bedingung formulieren:

```
(zeichen >= '0') AND (zeichen <= '9')
```

Mit dem Mengenoperator »IN« kann dies sehr viel einfacher bewerkstelligt werden:

```
zeichen IN ['0'..'9']
```

Auch in Verbindung mit der »CASE«-Anweisung kann eine Menge besonders nützlich sein. Wir erinnern uns, daß in der »CASE«-Anweisung sämtliche Werte, die die untersuchte Variable annehmen kann, berücksichtigt werden müssen. Sollen nur bestimmte Fälle ausgewertet werden, so empfiehlt es sich, durch eine vorangestellte IF-Anweisung vorher zu prüfen, ob der Wert einer der zu unterscheidenden ist, wie es das Beispiel zeigt:

```
IF farbe IN [rot...blau] THEN CASE farbe OF
                   rot: WRITE('Rot');
                  gelb: WRITE('Gelb');
                 gruen: WRITE('Gruen');
                  blau: WRITE(Blau')
                  END;
ELSE WRITE('Diese Farbe kenne ich nicht.');
```

Die Anweisung nach »ELSE« wird ausgeführt, wenn keiner der Fälle auf den Wert von »farbe« zutrifft. Weitere Möglichkeiten der Manipulation, außer mit den genannten Operatoren, sind in Pascal leider nicht möglich. Zur komfortableren Verarbeitung von Mengen wird sich der Programmierer selbst Prozeduren und Funktionen schreiben müssen, um die Möglichkeiten des Mengentyps voll auszunutzen. Auch die bei jedem Compiler unterschiedliche Maximalanzahl von Elementen in einer Menge läßt sich mit einem Kniff umgehen. Man definiert eine Mengengröße einfach als Array von kleinen Teilmengen.

Operator	Bedeutung	Operanden	Ergebnistyp
+	Vereinigung	Mengentyp	Mengentyp
*	Durchschnitt	Mengentyp	Mengentyp
-	Differenz	Mengentyp	boolean
<=	Inklusion	Mengentyp	boolean
>=	(ist Teilmenge von)	Mengentyp	boolean
=	Gleichheit	Mengentyp	boolean
<>	Ungleichheit	Mengentyp	boolean
IN	Element von	einfach und Mengentyp	boolean

Bild 13. Mengenoperatoren

Um eine feste Anzahl von Komponenten (die verschiedene Typen untereinander haben können) zusammenzufassen, gibt es in Pascal den Record-Typ. Er wird auch Verbundtyp genannt, da er eine Anzahl verschiedener Datentypen zu einem Verbund zusammenschließt. Die verschiedenen Komponenten werden auch »Felder eines Records« bezeichnet. Die Definition eines Record-Typs geschieht mit dem Schlüsselwort »RECORD«, das von einer Liste von Feldnamen sowie, mit einem Doppelpunkt getrennt, dem dazugehörigen Datentyp gefolgt ist. Die Definition muß mit »END;« abgeschlossen werden:

```
RECORD (Liste von Feldnamen):(Typen) END;
```

Will man zum Beispiel die zusammengehörigen Daten seiner Lieblingsschallplatte verarbeiten, so kann man dies mit Hilfe eines Records tun. Den Oberbegriff des Records nennen wir dabei »lieblingsplatte« und die einzelnen Felder, die verwendet werden, »stil«, »jahr« und »preis«. Die Definition hierfür sieht wie folgt aus:

```
TYPE schallplatte = RECORD
                    stil: (unterhaltung,klassik,
                    rock,pop);
                    jahr: 1970..1986;
                    preis: 10..20;
                    END;
```

Der Datentyp »schallplatte« besteht damit aus dem Feld »stil«, das vier Stilrichtungen beinhalten darf. Die zweite Feldkomponente ist das »jahr«, das Werte von 1970 bis 1986 annehmen kann, und schließlich das Feld »preis« mit dem Wertebereich von 10 bis 20. Durch die Variablen-Deklaration

```
VAR lieblingsplatte: schallplatte;
```

wird schließlich der Record »lieblingsplatte« definiert. Wie Sie sehen, kann ein Record Komponenten verschiedenen Typs aufnehmen. Die Komponenten eines Records werden durch den Variablennamen, gefolgt von einem Punkt (»record selector«) und dem jeweiligen Feldnamen beschrieben. So hat der eben definierte Record folgende Komponentennamen:

```
lieblingsplatte.stil
lieblingsplatte.jahr
lieblingsplatte.preis
```

Diesen Record-Namen können jetzt ihrem Typ entsprechende Daten zugewiesen werden:

```
lieblingsplatte.stil := rock;
lieblingsplatte.jahr := 1984;
lieblingsplatte.preis := 18;
```

Auch folgende Anweisung ist korrekt:

```
IF lieblingsplatte.preis > 17 THEN WRITE
('zu teuer')
ELSE WRITE('preisgünstig');
```

Es ist (wie bei den anderen strukturierten Datentypen auch) möglich, mehrere Records ineinander zu schachteln, das heißt, es kann ein Zugriff auf Felder von Feldern erfolgen. Dadurch ergibt sich eine mehrfache Selektion. Definieren wir zum Beispiel folgendes Record:

```
VAR a1: RECORD
         a2: RECORD
              a3: integer;
              a4: char;
              END;
         a5: real
        END;
```

Die selektierte Variable »a1.a2« ist dabei wieder selbst ein Record mit den Feldern »a3« und »a4«. Die Variable »a1.a2.a3« ist damit eine Variable des Typs integer. Die selektierte Variable »a1.a2.a4« hat dagegen den Typ char, da das Record-Feld »a4« als char definiert wurde. Die Variable »a1.a5« wurde nicht weiter selektiert, sondern erhielt gleich den Typ real. Somit sind folgende Anweisungen korrekt:

```
a1.a2.a3 := 10000;
a1.a2.a4 := 'r';
a1.a5 := 3.14157;
```

Außer dem Zuweisungsoperator gibt es in Pascal keine speziellen Operatoren, Funktionen oder Prozeduren, die Record-Variablen verarbeiten. Insbesondere die Ein- und Ausgabe ist mit den normalen Standardprozeduren nur begrenzt möglich. Während Record-Variablen der vier Standardtypen die »WRITE«- und »READ«-Anweisungen verwenden können, liegt die Ein- und Ausgabe von Record-Variablen des Aufzählungstyps zum Beispiel in der Kunst des Programmierers, der dafür eigene Prozeduren schreiben muß. So kann das obige Record-Beispiel bei der selektierten Variable »lieblingsplatte.preis« die folgende Anweisung verwenden:

```
WRITE(lieblingsplatte.preis);
```

Nicht gestattet ist dagegen die Ausgabe der Record-Variablen »lieblingsplatte.stil«, da diese vom Aufzählungstyp ist.

```
WRITE(lieblingplatte.stil);
```

ist demnach falsch.

Wie bei den anderen Datentypen können auch Arrays aus Records gebildet werden. Dies ist nützlich, wenn man eine große Anzahl an gleichartigen Records benötigt. Hat man zum Beispiel seine Plattensammlung zu verwalten, so kann man ein Array mit mehreren gleichen Records definieren. Unter der Voraussetzung, daß die obige Typendefinition von »schallplatte« noch gilt, kann man folgende Variablen-Deklaration machen:

```
VAR platte: ARRAY [1..100] OF schallplatte;
```

Der Variablenname »platte« bestimmt nun ein Array von hundert gleichartigen Records, die alle die Daten für eine Schallplatte der Sammlung repräsentieren sollen. Die einzelnen Komponenten werden dabei unter Angabe des jeweiligen Index unterschieden. So ist

```
platte[1].stil
```

eine Record-Komponente des ersten Arrays, die in unserem Fall die Stilrichtung der ersten Platte unserer Sammlung enthalten soll.

```
platte[50].preis
```

bezieht sich auf den Preis von Platte Nummer 50. Man sieht, daß Records dabei helfen, problemorientierter zu programmieren. Insbesondere bei der Verwaltung von großen Datenmengen, die untereinander zwar unterschiedlich sind, sich aber immer wiederholen, können Records sehr nützlich sein. So könnte die Mitgliederliste eines Vereins beispielsweise in Pascal recht einfach gestaltet werden. Listing 17 zeigt die Eingabe der Daten verschiedener Personen in Records.

Für mehrere Anweisungen, die sich immer wieder auf die Felder desselben Records beziehen, gibt es in Pascal eine Anweisung, die es erlaubt, durch eine einmalige Angabe des Record-Namens mehrere gleichartige Record-Anweisungen zusammenzufassen.

Die »WITH«-Anweisung hat folgenden Syntax:

```
WITH (Record-Variable) DO (Anweisung);
```

Die Anweisung hinter »DO« kann auch eine Verbundanweisung sein, die, wie bekannt, von »BEGIN« und »END« umgeben sein muß. Durch Angabe der Record-Variablen genügt es bei »WITH« (bei Bezugnahme auf die einzelnen Felder des Records), nur deren Feldnamen anzugeben. So kann die Ausgabe der Felder eines Records sehr einfach geschehen, da nicht bei jeder Anweisung der Record-Name vor den Feldnamen einer jeder Variablen geschrieben werden muß. Die Anweisung

```
WITH lieblingsplatte DO BEGIN
    WRITE(jahr);
    WRITE(preis)
END;
```

gibt die Werte der Record-Komponenten »lieblingsplatte.jahr« und »lieblingsplatte.preis« aus und ist somit äquivalent zu folgenden Anweisungen:

```
WRITE(lieblingsplatte.jahr);
WRITE(lieblingsplatte.preis);
```

Die Anwendungsmöglichkeiten von Records sind sehr umfangreich, so daß sie im Rahmen dieses Kurses, wie die

```
(* TEILNEHMER-RECORDS EINLESEN, NACH DER NUMMER SORTIEREN,
   UND WIEDER AUSGEBEN *)
PROGRAM NUMMERN;
  TYPE TEILNMRTYP=
    RECORD
      NAME: PACKED ARRAY [1..18] OF CHAR;
      ADRESSE: PACKED ARRAY[1..23] OF CHAR;
      TELEFONNUMMER: PACKED ARRAY[1..8] OF CHAR
  END;
VAR ARBEITSBEREICH: TEILNMRTYP;
    TEILNEHMER: ARRAY[1..25] OF TEILNMRTYP;
    I, J, RECORDANZ: INTEGER;
    CH: CHAR;

PROCEDURE SORTIERE(RECORDANZ: INTEGER);
  (* SORTIERE RECORDS NACH DER TELEFONNUMMER *)
  VAR ARBEITSBEREICH: TEILNMRTYP;
      I, J: INTEGER;
  BEGIN
    FOR I:=1 TO RECORDANZ-1 DO
      FOR J:=1 TO RECORDANZ-1 DO
        IF TEILNEHMER[J+;].TELEFONNUMMER >
                TEILNEHMER[J+1].TELEFONNUMMER THEN
           BEGIN
        (* VERTAUSCHEN VON TEILNEHMER[J] UND TEILNEHMER[J+1] *)
             ARBEITSBEREICH:=TEILNEHMER[J];
             TEILNEHMER[J]:=TEILNEHMER[J+1];
             TEILNEHMER[J+1]:=ARBEITSBEREICH
           END
  END;
BEGIN
  WRITE('ANZAHL DER RECORDS = ');
  READLN(RECORDANZ);
  (* EINLESEN DER RECORDS IN DEN ARRAY *)
  WRITELN('NAME', 'ADRESSE':21, 'TELEFON':23);
  FOR I:=1 TO RECORDANZ DO
    BEGIN
      FOR J:=1 TO 18 DO
        BEGIN
          READ(CH);
          ARBEITSBEREICH.NAME[J]:=CH
        END;
      FOR J:=1 TO 23 DO
        BEGIN
          ARBEITSBEREICH.ADRESSE[J]:=CH
        END;
      FOR J:=1 TO 8 DO
        BEGIN
          READ(CH);
          ARBEITSBEREICH.TELEFONNUMMER[J]:=CH
        END;

      TEILNEHMER[I]:=ARBEITSBEREICH;
      READLN
    END;
  (* SORTIERE DIE RECORDS NACH DEN TELEFONNUMMERN *)
  SORTIERE(RECORDANZ);
  (* GIB DEN SORTIERTEN ARRAY AUS RECORDS AUS *)
  WRITELN;
  WRITELN('DIE LISTE WURDE NACH TELEFONNUMMERN SORTIERT');
  WRITELN('TELEFON', 'NAME':6, 'ADRESSE':21);
  FOR I:=1 TO RECORDANZ DO
    WRITELN(TEILNEHMER)[I].TELEFONNUMMER,
      ' ', TEILNEHMER[I].NAME, TEILNEHMER[I].ADRESSE)
END.
```

Listing 17. Records

anderen Datentypen, eigentlich nur relativ kurz behandelt werden konnten. Insbesondere bei Verschachtelungen der einzelnen Datentypen untereinander lassen sich sehr komplexe Strukturen aufbauen. Die gesamten Möglichkeiten von strukturierten Datentypen aufzuzählen, würde wohl einige Sonderhefte füllen. Um jedoch auch den letzten der Datentypen zu erwähnen, kommen wir jetzt zu dem Typ File, der insbesondere für die Arbeit mit Externspeichern, wie zum Beispiel einem Diskettenlaufwerk, gedacht ist.

Dateien auf Diskette

Files sind in Pascal eine beliebig lange Folge von Komponenten des gleichen Typs. Die Länge darf dabei während des Programmablaufs variieren. Der Zugriff auf eine Komponente eines Files erfolgt jedoch auf eine ganz andere Art als bei den übrigen Datentypen. Anders als bei Records oder Arrays, die eine direkte Auswahl von Komponenten (durch Index oder Feldname) erlauben, geschieht eine Bearbeitung von Filedaten sequentiell, das heißt immer der Reihe nach. Diese Art der Datenverarbeitung ist zum Beispiel auch auf Diskettenlaufwerken realisiert, die meist nur sequentiell Daten lesen und speichern können. Eine Besonderheit ist, daß ein Pascal-File entweder nur ausgelesen oder nur beschrieben werden kann. Bei wechselndem Schreib- und Lesezugriff muß zuerst auf Schreiben beziehungsweise Lesen umgeschaltet werden. Bei der Ein- und Ausgabe am Bildschirm haben wir schon unbewußt mit Files zu tun gehabt. Dies waren die Standarddateien »input« und »output«, die für die Eingabe von der Tastatur und für die Ausgabe auf dem Bildschirm verantwortlich sind. Diese Files sind von UCSD-Pascal standardmäßig vorgegeben und mußten deshalb nicht extra im Programm angegeben werden. Andere Files, auf der Diskette zum Beispiel, müssen im Programmkopf angegeben werden. Dies geschieht durch den Namen der Datei, der hinter dem Programmnamen in runden Klammern geschrieben wird. Es soll nun der File-Typ beschrieben werden. Er wird wie folgt formuliert:

```
FILE OF (Typ)
```

Bei (»Typ«) kann es sich um jeden beliebigen Typ, insbesondere auch um strukturierte Typen, handeln. Bei einem File sind nicht alle Daten gleichzeitig und von jeder Stelle abrufbar, es ist vielmehr immer nur das Element verfügbar, das an der aktuellen File-Position steht. Um dies zu veranschaulichen, wollen wir den Sachverhalt in einem Beispiel demonstrieren. Wir deklarieren eine Variable mit dem Namen »e« als File. Wir können dabei das Schlüsselwort »FILE« direkt in der Variablen-Deklaration schreiben:

```
VAR e: FILE OF y
```

Der Typ »y« muß selbstverständlich vordefiniert worden sein. Er kann, wie schon gesagt, beliebig sein. Zusammen mit der Deklaration wird gleichzeitig eine sogenannte Puffervariable geschaffen, die den gleichen Namen wie unser File hat, aber von einem Pfeil gefolgt ist. Sie heißt »e^« und enthält immer den aktuellen Wert der gerade verfügbaren File-Komponente. Man kann sich »e^« wie ein Fenster vorstellen, in dem sich immer die aktuelle Komponente befindet. Bild 14 stellt dies grafisch dar.

Man kann also nur durch Lesen oder Schreiben der Puffervariablen einen Zugriff auf das File vornehmen. Es gibt allerdings noch weitere Funktionen, die das Fenster an den Anfang zurücksetzen oder um eine Komponente weiter, bewegen. Die Standardfunktion »EOF« (»end of file«) hat hier eine Bedeutung: sie zeigt das Ende eines Files an. In unserem Fall hat »EOF(e)« den Wert »true«, wenn das Fenster (»e^«) am Schluß des Files steht und keinen Wert mehr enthält (Ende des Files).

Zum Rücksetzen der Puffervariablen auf die erste Komponente des Files muß die Standardprozedur »RESET()« verwendet werden.

```
RESET(e);
```

setzt das Fenster also auf den Beginn des Files »e«. »EOF(e)« ergibt den Wert »true«, wenn das File leer ist. Beim nicht leeren File wird »EOF(e)« »false«, während »e^« den ersten Wert des Files enthält. Gleichzeitig wird es in den Lesezustand gebracht, das heißt, es kann nur gelesen werden. Das Lesen einer Komponente des Files »e« geschieht durch die Prozedur »GET(e)«. Man wendet sie folgendermaßen an:

```
v := e^;  GET(e);
```

Die Variable »v«, die natürlich den gleichen Typ wie die Komponenten des Files haben muß, bekommt den Wert der aktuellen Komponente zugewiesen. Die Prozedur »GET()« bewirkt nun, daß das Fenster eine Position weitergeschoben wird und die Puffervariable »e^« den Wert der neuen Komponente annimmt.

Wird die Prozedur »GET()« am Ende des Files angewendet (»EOF(e)« ist »true«), so kann »e^« kein neuer Wert mehr zugewiesen werden. Die meisten Pascal-Compiler akzeptieren das nicht und brechen den Programmablauf ab. Durch die Prozedur »REWRITE()« können Daten in das File geschrieben werden.

```
REWRITE(e);
```

ermöglicht dies für unser Beispielfile. Das Problem dabei ist, daß bei Anwendung dieser Prozedur das gesamte File gelöscht wird, um dann erst in den Schreibzustand umzuschalten. Die Puffervariable zeigt dann auf den Anfang des Files und es kann mit dem Schreiben begonnen werden. Dies geschieht mit der ebenfalls standardmäßig vorgegebenen Prozedur »PUT()«. Es ist folgende Befehlsfolge nötig:

```
e^ := v;  PUT(e);
```

Dieses Mal wird der Puffervariablen »e^« der zu schreibende Wert zugewiesen. Er muß selbstverständlich zum Typ des Files passen. Die Put-Prozedur schreibt den Inhalt von »e^« an die aktuelle Position und schiebt das Fenster eine Position weiter. »EOF(e)« bleibt bei »PUT(e)« immer »true«, da das Fenster ja schon auf der nächsten leeren Position ist. Bild 15 zeigt den Schreibvorgang in einer Grafik.

Dadurch, daß das File vor dem Umschalten auf Schreiben ganz gelöscht wird, ist das Einfügen oder Verändern der Werte im File sehr umständlich. Eine Möglichkeit ist, die Komponenten des Files bis zur Einfügestelle in ein Hilfsfile zu kopieren, den neuen oder veränderten Wert an das Hilfsfile anzuhängen und schließlich den Rest des Files dahinter zu schreiben. Danach kann das gesamte Hilfsfile wieder in das Ausgangsfile zurückkopiert werden. Diesen Vorgang sollte man durch eigene Prozeduren abwickeln.

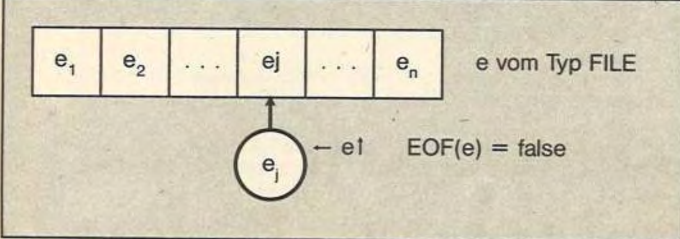


Bild 14. Puffervariable »e«

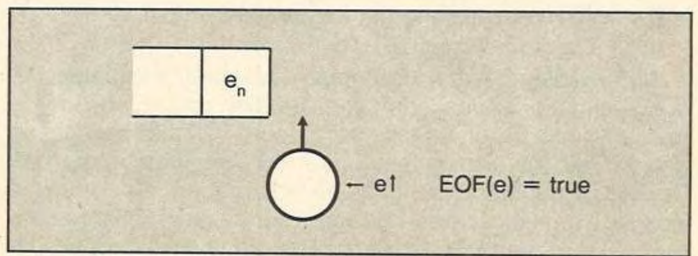


Bild 15. Neuer Wert für »e«

Ein Text ist in Pascal eine Folge von Zeichen, also eine Zeichenkette. Er ist der Definition von string sehr ähnlich. So kann man einen neuen Datentyp folgendermaßen definieren:

```
TYPE text = FILE OF char;
```

»text« ist damit ein File-Typ, der Druckzeichen aufnehmen kann. Mit ihm können wir ein File eröffnen, welches das Speichern von Texten sehr einfach macht. Dieser Typ ist von Pascal jedoch schon vordefiniert und kann somit auch ohne die vorhergehende »TYPE«-Anweisung benutzt werden. Die bereits angesprochenen Standardnamen »input« und »output« sind zum Beispiel Files des Typs »text«. Man kann damit aber auch eigene Textfiles, beispielsweise auf Diskette, definieren. Mit der einfachen »VAR«-Anweisung ist dies schnell getan:

```
VAR brief: text;
```

Die Variable »brief« kann nun als Textfile verwendet werden. Es ist möglich, ein Textfile mit den bekannten Prozeduren »GET()« und »PUT()« zu bearbeiten. Wir erinnern uns, daß die Textfiles »input« beziehungsweise »output« die Eingabe über die Tastatur beziehungsweise die Ausgabe auf dem Bildschirm übernommen haben. Wir haben zu diesem Zweck die Prozeduren »GET()« und »PUT()« jedoch nie verwendet, sondern die komfortablen Prozeduren »READ()« und »WRITE()«. Diese Anweisungen sind Erweiterungen der Prozeduren »GET()« und »PUT()« und erlauben, wie Sie ja schon wissen, das Beschreiben und das Lesen eines Textfiles auf recht angenehme Weise. Die Eingabe von der Tastatur und die Ausgabe auf dem Bildschirm war demnach nichts weiter als das Lesen der Datei »input« und das Beschreiben der Datei »output«. Die Prozeduren »READ()« und »WRITE()« sind jedoch allgemein für Textfiles bestimmt und können deshalb auch für eigene Files verwendet werden. Das umständliche Herumhantieren mit »GET()« und »PUT()« ist dadurch nicht mehr notwendig. Durch Angabe des File-Namens in der Anweisung wird das Lesen und Schreiben auf dieses File bezogen. Das Lesen eines Textfiles kann folgendermaßen ablaufen:

```
READ(brief,w1,w2,w3);
```

Damit wird das File »brief« angesprochen, das wir oben als Beispiel hatten. Sie werden sich fragen, warum man bei der Tastatureingabe das File »input« nicht in der »read«-Anweisung angeben muß. Dies liegt am Pascal-Compiler selbst. Sollte der File-Name fehlen, so wird der Befehl automatisch auf die Standardatei »input«, also auf die Tastatur, bezogen. Die Anweisung

```
READ(input,w1,w2,w3,...,wn);
```

kann also vereinfacht so geschrieben werden:

```
read(w1,w2,w3,...,wn);
```

Bei »WRITE()« wird dies gleichermaßen gehandhabt. Das Schreiben auf ein Textfile kann durch

```
WRITE(brief,w1,w2,w3,...,wn);
```

geschehen. Auch hier ist das Textfile »brief« wieder ein Beispiel. Bei der Ausgabe auf dem Bildschirm darf, analog zu »READ()«, der Filename »output« auch weggelassen werden.

```
WRITE(output,w1,w2,w3,...,wn);
```

ist demnach äquivalent zu der bekannten Anweisung

```
WRITE(w1,w2,w3,...,wn);
```

Auch die Standardfunktionen »EOF()« und »EOLN()« verhalten sich nach diesen Regeln:

EOF(input) ist äquivalent zu EOF
EOLN(input) ist äquivalent zu EOLN

So können sämtliche »PUT«- und »GET«-Anweisungen durch »WRITE« und »READ« ersetzt werden. Für ein File »f« gilt also:

```
READ(f,x)     ist gleich den Anweisungen x := f^;
                                          GET(f);
WRITE(f,x)    ist gleich den Anweisungen f^ := x;
                                          PUT(f);
```

Jetzt wäre es schön, wenn »READ()« und »WRITE()« auf Files aller Datentypen angewandt werden könnten. Dem ist auch so. Die Prozeduren »READ()« und »WRITE()« sind so flexibel, daß sie auch Files anderer Typen zulassen. Diese Typen können insbesondere auch strukturiert sein, wie zum Beispiel Arrays und Records. Die Bestimmung, auf welchem Datenträger ein File behandelt werden soll, ist von Implementation zu Implementation unterschiedlich. Informationen dazu sind dem jeweiligen Handbuch zu entnehmen. Kommen wir nun zu den Zeigerstrukturen.

Zeiger als Variable

Mit »Pointern« wird eine vollkommen neue Art von Variablen eingeführt. Sie sind nicht wie andere Typen an die Blockstruktur des Programmes gebunden, sondern können dynamisch erzeugt und wieder desaktiviert werden. »Pointer« ist der englische Ausdruck für Zeiger, womit die Bedeutung schon halbwegs klar wird. Diese Zeiger weisen auf eine Stelle hin, an der sich ein bestimmter Wert befindet. Der Pointer ist dabei selbst wieder ein Wert, man muß ihn aber von dem Wert, auf den er zeigt, streng unterscheiden. Ein Zeiger hat in Pascal auch einen bestimmten Bezugstyp. Dieser gibt an, auf welche Art von Werten er zeigen darf. Ein Zeigertyp wird durch den nach oben gerichteten Pfeil und einen beliebigen Datentypen definiert:

```
^(Typ)
```

So ist die Vereinbarung

```
TYPE zeigertyp = char;
     zeiger = ^zeigertyp;
```

eine Definition, die den Typ »zeiger« als Zeigertyp des Bezugstyps char bestimmt. Eine Variable, die den Typ »zeiger« erhält, wie zum Beispiel

```
VAR z: zeiger;
```

nennt man »Zeigervariable«. Eine Variable dieses Typs kann nun als mögliche Werte Zeiger aufnehmen, die nur auf Werte des Typs char zeigen dürfen. Jeder Zeiger kann auch den Wert des »leeren« Zeigers haben, das heißt einem Zeiger, der auf keinen Wert zeigt. Er hat dann nicht den Wert Null, sondern den speziellen Zeigerwert »NIL« (deutsch: nichts). Man kann sich einen Zeiger im Computer als einen Wert vorstellen, der eine bestimmte Adresse im Speicher darstellt. An dieser Adresse steht der Wert, auf den er zeigen soll. Haben wir eine Zeigervariable wie »z« definiert, so ist dies noch keine Variable des Bezugstyps, in unserem Fall des Typs char. »z« hat auch noch keinen Wert, der auf eine Speicherzelle zeigt, nicht einmal den Wert »NIL«. Er muß also erst, wie andere Variablen auch, mit einem Wert belegt werden. Daher muß zuerst Speicherplatz für einen Wert bereitgestellt werden, auf den er dann zeigen kann. Dies kann durch die Standardprozedur »NEW()« geschehen:

```
NEW( (Zeigervariable) );
```

Durch Angabe des Namens der Zeigervariablen wird mit »NEW()« ein Speicherplatz reserviert, der es erlaubt, einen Wert des Bezugstyps aufzunehmen. In unserem Beispiel wird mit

```
NEW(z);
```

der Speicherplatz für einen Wert des Typs char eingerichtet. Er kann genau ein Druckzeichen aufnehmen. Der Zeiger »z« erhält nun einen Wert, nämlich den Zeigerwert auf diese Speicherstelle. Der Speicherplatz selbst hat dabei keinen eigenen Namen, sondern kann nur über den Zeiger »z« erreicht werden. Der Wert des Speicherplatzes ist durch den Zeigernamen, gefolgt von einem Pfeil (»^«), ansprechbar.

```
z^
```

Man nennt diese Angabe auch »Bezugsvariable«, da durch sie Bezug auf den reservierten Speicherplatz genommen werden kann. Damit kann man den Wert der Speicherstelle auslesen oder beschreiben. So ist die Anweisung

```
z^ := 'a';
```

korrekt. Dem Zeigernamen selbst darf aber kein Wert des

Bezugstyps zugewiesen werden (er steht ja nicht für eine Variable dieses Typs).

```
z := 'z';
```

ist somit nicht zulässig. Die Zuweisung von

```
z^ := 'a'
```

ist in Bild 16 schematisch dargestellt worden.

Eine erneute »NEW«-Anweisung mit dem selben Zeiger erzeugt einen weiteren Speicherplatz, der einen Wert des Bezugstyps aufnehmen kann. Der alte Wert wird dabei nicht gelöscht, kann aber nicht mehr angesprochen werden, da der Zeiger jetzt auf die neue Speicherstelle weist. Der alte Wert kann zum Beispiel ansprechbar bleiben, wenn man zuvor einem anderen Zeiger den Zeigerwert überträgt. So läßt die zusätzliche Vereinbarung

```
VAR hilfe: zeiger;
```

folgende Zuweisung zu:

```
hilfe := z;
```

Damit zeigt nun auch »hilfe« auf den Wert, auf den auch »z« weist. Durch erneutes Anwenden von »NEW(z)« wird ein weiterer Speicherplatz reserviert, auf den »z« jetzt zeigt. Der ursprünglich Wert von »z« geht aber dadurch nicht verloren, da durch »hilfe« der alte Zeigerwert festgehalten wurde. Um einen Wert endgültig zu löschen, muß die Standardprozedur »DISPOSE()« benutzt werden. Sie gibt den reservierten Speicherplatz wieder frei. Durch

```
DISPOSE(z);
```

würde unser neu geschaffener Speicherplatz, auf den »z« zeigt, wieder desaktiviert werden. Der darin gespeicherte Wert geht dadurch ebenfalls verloren. Allgemein kann man sagen, daß eine Bezugsvariable, die durch »NEW()« erzeugt wurde, solange erhalten bleibt, bis sie durch »DISPOSE()« oder das Programmende wieder außer Kraft gesetzt wird. Die Frage stellt sich nun, was man mit solchen Zeigern und den sogenannten dynamischen Bezugsvariablen anfangen kann.

Zeiger auf Records

Durch Überlegung dürften Sie vielleicht schon auf die Idee gekommen sein, daß durch mehrmaliges Ausführen von »NEW()« eine in gewisser Weise zusammenhängende Struktur von Werten erzeugt werden könnte. Dies ist auch möglich, wenn wir als Bezugstyp einen der strukturierten nehmen. Damit diese Verkettungen außer Zeigern auch andere Informationen aufnehmen können, ist die Datenstruktur von Records geeignet. Wir wählen einen Record, der aus zwei Feldkomponenten besteht. Ein Feld nimmt die Informationen auf, die verkettet werden sollen, das andere ist ein Zeigertyp, der als Bezugstyp gerade diesen eben besprochenen Record hat. In Pascal-Formulierungen sieht dies so aus:

```
TYPE typ = RECORD
           data: integer;
           naechster: ^typ
           END;
```

Die Feldkomponente des Recordtyps »typ«, die Daten aufnehmen soll, heißt »data«, die Zeigerkomponente hat den Namen »naechster« und ist wiederum vom Zeigertyp »typ«. Man sieht, daß auch in der Typendefinition Rekursivität auftreten kann. Das Besondere ist, daß hierbei eine Typendefinition schon verwendet werden darf, noch bevor sie komplett definiert wurde. Wir erinnern uns, daß alle Namen, auch Typennamen, bisher nur verwendet werden konnten, wenn sie schon zuvor definiert wurden. Ist die Typendefinition jedoch rekursiv wie in unserem Fall, muß diese Reihenfolge nicht eingehalten werden. Die Voraussetzung dafür ist aber, daß der Aufruf des Typennamens in seiner eigenen Typendefinition auftritt. Ansonsten ist dies nicht gestattet. Doch nun zurück zu unserem Problem. Nachdem »typ« vereinbart wurde, kann eine Variable als Zeiger dieses Typs deklariert werden:

```
VAR z: ^typ;
```

Die Variable »z« ist nun eine Zeigervariable. Die Anweisung

```
NEW(z);
```

ist also korrekt. Durch »NEW(z)« wird Speicherplatz geschaffen, der die Feldkomponenten von »typ« aufnehmen kann. Die beiden Komponenten können nun durch die Bezugsvariable »z^« des Zeigers »z« aufgerufen werden. So gibt es die Komponente »z^.data«, die die Information aufnehmen soll, und die Komponente »z^.naechster«, die wiederum ein Zeiger ist, nämlich vom Typ »^typ«. Damit kann »z^.naechster« wieder für »NEW()« verwendet werden. Rufen wir also »NEW()« nochmals auf:

```
NEW(z^.naechster);
```

»z^.naechster« zeigt nun als Komponente des zuerst erstellten Records auf das neu geschaffene Record. Eine neue Bezugsvariable mit dem Namen »z^.naechster^« ist entstanden. Sie hat als Komponenten wieder »data« und »naechster«. Das Record-Feld »naechster« ist aber wiederum ein Zeiger des Typs »typ«. So ist also

```
z^.naechster^.naechster
```

ein weiterer Zeiger. Analog dazu könnte man jetzt mit diesem Zeiger wieder die Prozedur »NEW()« aufrufen, um einen neuen Zeiger zu schaffen, der einen nächsten Zeiger erzeugen könnte. Das Ergebnis ist also eine Kette von Daten, welche in den jeweiligen Komponenten »data« stehen. Sie hängen durch die sich selbst erzeugenden Zeiger zusammen, indem jeder vorhergehende Zeiger auf den nächsten Record zeigt. Der Nachteil bei dieser Methode ist aber, daß man durch die Rekursion immer längere und neue Komponentennamen erhält. Somit muß auch die »NEW«-Anweisung für jeden Zeiger neu geschrieben werden. Diese Art der Verkettung ist also nicht gerade praktisch. Es gibt jedoch eine andere Methode, um unser Problem in den Griff zu bekommen. Die Lösung wurde bereits zu Beginn des Abschnitts über Zeiger teilweise dargelegt. Das Geheimnis ist das »Umhängen« von Zeigern. Listing 18 stellt eine Verkettungsmethode durch Umhängen dar. Es wurden dabei die gleichen

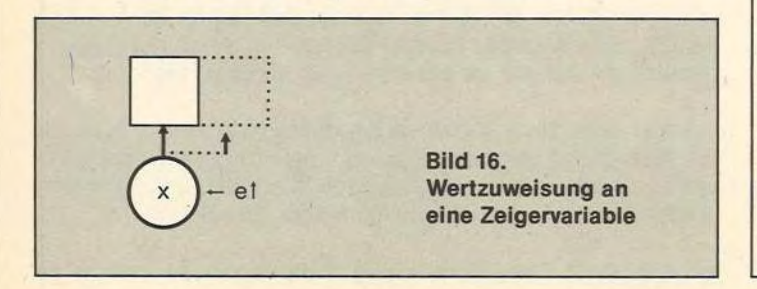


Bild 16. Wertzuweisung an eine Zeigervariable

```
PROGRAM liste;
 TYPE typ = RECORD
            data: integer;
            naechster: ^typ;
           END;
 VAR z,h: ^typ;
   BEGIN
     h := NIL;
     WHILE NOT eof DO
          BEGIN
            NEW(z);
            READLN(z^.data);
            (* Umhaengen *)
            z^.naechster := h;
            h := z;
          END;
END.
```

Listing 18. Verkettete Listen durch Umhängen von Zeigern

Variablen- und Typnamen wie im obigen Beispiel verwendet. Es wird lediglich ein weiterer Zeiger »h« des gleichen Typs wie »z« benötigt. Er ist zusätzlich deklariert worden. Zu Beginn des Programms bekommt der Zeiger »h« den Wert »NIL« (leerer Zeiger), da noch kein Element in der Liste vorhanden ist. Die Schleife beginnt. Mit

```
NEW(z);
```

wird Speicherplatz für die Bezugsvariable »z^« geschaffen. »z« zeigt nun auf diese. Danach wird die Information in die Datenkomponente »z^.data« des Records gelesen. Nun geschieht das geheimnisvolle Umhängen. Mit

```
z^.naechster := h;
```

wird der Zeigerwert von »h«, der am Anfang noch »NIL« ist, der Zeigerkomponente, die durch »NEW()« entstanden ist, zugewiesen. Danach wird

```
h := z;
```

ausgeführt, was bedeutet, daß der Zeigerwert des Zeigers »h« den Wert von »z« bekommt. »h« zeigt somit, wie im Moment noch »z«, auf das erste Element unserer Liste. Lassen wir die Schleife nun ein zweites Mal durchlaufen. Durch erneutes Aufrufen von »NEW(z)« wird ein neuer Speicherplatz reserviert, auf den »z« jetzt zeigt. Wir erinnern uns, daß der Zeigerwert von »z« durch das Umhängen nicht verlorenging! Er steht jetzt in »h«. Die Feldkomponente »z^.data«, die jetzt eingelesen wird, hat mit der vorherigen nichts zu tun, da sich der Name durch »NEW()« auf eine andere Speicherstelle bezieht. Ein erneutes Umhängen beginnt. Auch »z^.naechster« wird an der neu geschaffenen Stelle gespeichert und ist somit ein neuer Zeiger. Er nimmt den Wert von »h«, der ja noch auf das erste Element zeigt. »z^.naechster« weist also auf ihm vorangehende Elemente der Liste. »h« nimmt nun den Wert des Zeigers »z« an, womit er auf das zweite Element zeigt. Analog dazu verlaufen auch die weiteren Schleifendurchgänge, die eine zusammenhängende Kette an Integer-Zahlen aufbauen. Die Zeiger der jeweiligen Elemente zeigen dabei immer auf das vorhergehende Element. Hat man eine Kette eingegeben, kann man diese zum Beispiel wieder auslesen. Wenn man dies der Reihe nach tut, muß man aber bedenken, daß bei unserer Verkettungsmethode das zuletzt eingegebene Element an oberster Stelle liegt und als erstes wieder ausgelesen werden muß. Diese Anordnung ist der des Stackregisters im Computer sehr ähnlich. Das behandelte Beispiel war natürlich sehr einfacher Art. Doch insbesondere bei komplizierteren Datentypen, wie zum Beispiel mehrmals geschachtelten Records, können solch verkettete Listen sehr praktisch sein. Dies dürfte aber eher zu den fortgeschrittenen Programmiertechniken gehören, die im Rahmen dieses Kurses aus Platzgründen nicht erklärt werden können. Sie dürften für Pascal-Neulinge auch zu komplex sein. Kommen wir lieber zum letzten Abschnitt unseres Streifzuges durch die Welt des strukturierten Programmierens. Er ist den Standardfunktionen und -prozeduren gewidmet, auf die noch kein Bezug genommen wurde. Dies sind im besonderen die Erweiterungen des UCSD-Pascals.

Erweiterter Standard

UCSD-Pascal hält sich weitestgehend an den Pascal-Standard, den N. Wirth geschaffen hatte. Doch wie am Anfang des Kurses schon gesagt wurde, hat der UCSD-Standard einige Erweiterungen, die die Textverarbeitung und die Programmierung von Grafik erleichtern sollen. Die Erweiterungen bestehen aus neuen Funktionen und Prozeduren, die standardisiert, das heißt in allen UCSD-Pascal-Implementationen bereits vordefiniert sind.

Die Funktionen für die Stringverarbeitung sind bei der Besprechung des Datentyps »string« bereits erwähnt worden. Eine Sammlung davon ist in Bild 8 zu sehen. Beginnen wir mit der Funktion »CONCAT«, die beliebig viele Strings zu einem einzigen zusammenfügt. Sie hat die Syntax:

```
CONCAT(string1,string2,string3,...,stringn)
```

So liefert beispielsweise die »CONCAT«-Anweisung

```
WRITELN(CONCAT('Dies ist ','ein
zusammengesetzter ','Satz.'));
```

die Ausgabe:

```
'Dies ist ein zusammengesetzter Satz.'
```

Im Gegensatz dazu kann mit »COPY« ein Teilstring aus einem anderen String kopiert werden. »COPY« ist folgendermaßen anzuwenden:

```
COPY(string,anfang,länge)
```

Die Werte »anfang« und »länge« müssen vom Typ integer sein und beinhalten die Anfangspositon und die Zeichenlänge des Teiles, der vervielfältigt werden soll. Beispiele hierfür sind:

```
WRITELN(COPY('Blaise Pascal',1,6));   ergibt
'Blaise'
WRITELN(COPY('Blaise Pascal',4,7));   ergibt
'ise Pas'
```

Man darf jedoch nicht mehr Zeichen kopieren, als ab der Anfangsposition bis zum Ende vorhanden sind.

```
WRITELN(COPY('Blaise Pascal',8,8));
```

verursacht eine Fehlermeldung, da ab Zeichenposition 8 nur noch 6 Zeichen stehen. POS() ist eine Testfunktion, die einen Wert des Typs integer liefert. Sie prüft, ob ein Musterstring in einem Quellstring enthalten ist. Ist das der Fall, so ist der Wert der Funktion die Zeichenposition, ab der das Muster mit dem Quellstring identisch ist. Ansonsten nimmt sie den Wert Null an. Einige Beispiele sollen dies verdeutlichen:

```
POS('und','hund')              ergibt den Wert 2
POS('spiel','kartenspiel')     ergibt den Wert 7
POS('butter','margarine')      ergibt den Wert 0
```

Für das Einfügen und Löschen von Strings in einen anderen String stellt UCSD-Pascal die Prozeduren »INSERT« und »DELETE« zur Verfügung. Die Syntax von INSERT lautet:

```
INSERT(string1,string2,anfang)
```

Damit wird String1 in den String2 ab der Position »anfang« eingefügt. Auch hier ein Beispiel:

```
wort := 'kein glueck';
INSERT('bisschen ',wort,6);
WRITELN(wort);
```

ergibt den Ausdruck:

```
kein bisschen glueck
```

Ebenso kann man Zeichen eines Strings löschen:

```
DELETE(string,anfang,anzahl)
```

Hier werden ab der Position »anfang« die Anzahl von »anzahl«-Zeichen aus dem String »string« gelöscht. Ein Beispiel:

```
wort := 'naturreligion';
DELETE(wort,4,7);
WRITELN(wort);
```

```
PROGRAM AUFSPALTEN;
  VAR ZEILE:STRING;
    POSITION:INTEGER;
  BEGIN
    WRITELN('TIPPEN SIE EINE TEXTZEILE EIN');
    READLN(Zeile);
    POSITION:=POS(' ',ZEILE);
    WHILE POSITION <> 0 DO
      BEGIN
        WRITELN(COPY(ZEILE,1,POSITION-1));
        DELETE(ZEILE,1,POSITION);
        POSITION:=POS(' ',ZEILE)
      END
    WRITELN(ZEILE)
  END.
```

Listing 19. Stringbehandlung

Das Ergebnis ist:

```
nation
```

Wie bei »COPY« ist es natürlich nicht möglich, mehr Zeichen zu löschen, als der String ab der angegebenen Anfangsposition hat. Dies hätte eine Fehlermeldung zur Folge. Um die Arbeit mit diesen Funktionsprozeduren zu verdeutlichen, ist in Listing 19 ein Programm abgedruckt, das »POS« und »DELETE« verwendet. Es spaltet einen eingegebenen Satz in die einzelnen Worte auf. Zu guter Letzt soll hier noch eine Pascal-Konstruktion erwähnt werden, die von Pascal-Programmierern, die etwas auf sich halten, möglichst vermieden werden. Aus diesem Grund wird sie auch am Schluß dieses Kurses behandelt. Es ist die aus Basic bekannte unbedingte Sprunganweisung oder »GOTO«-Anweisung, die in Pascal allen Regeln trotzt. Die Vorteile des strukturierten Programmierens können damit zunichte gemacht werden. Sprünge in Pascal können nur auf vordefinierte Marken geschehen, da ja keine Zeilennummern zur Verfügung stehen. Marken, die im Programm vorkommen, müssen im Deklarationsteil angegeben werden. Dies geschieht durch das Schlüsselwort »LABEL«:

```
LABEL (Marken);
```

Marken dürfen in Pascal nur aus natürlichen Zahlen bestehen, wie zum Beispiel

```
111,5,0815
```

Falsche Marken wären dagegen

```
lab3,-100,marke
```

Im Anweisungsteil des Programmes können dann diese Marken auftauchen. Wird eine Marke angesprungen, so wird die Anweisung, die (mit einem Doppelpunkt getrennt) hinter der Marke steht, ausgeführt. Eine Marke darf nur eine Anweisung markieren, kann aber beliebig oft angesprungen werden. Marken im Anweisungsteil sind zum Beispiel

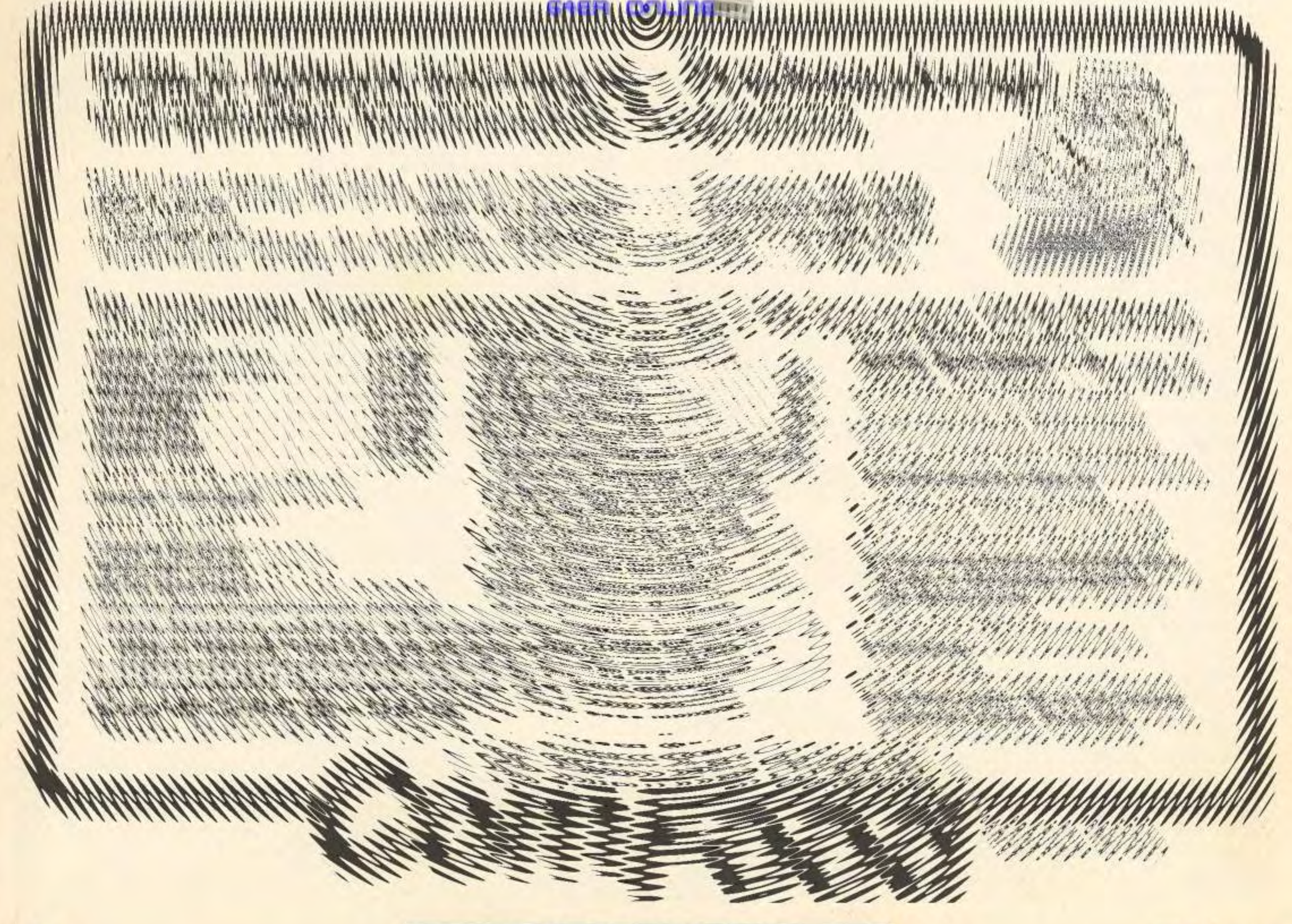
```
10: x := sqr(x);    5: WRITE('Label 5');
```

Sie können mit dem Statement »GOTO« angesprungen werden.

```
GOTO 10;
```

bewirkt etwa einen Sprung auf die Marke 10. Der Gültigkeitsbereich von Marken ist analog zum Gültigkeitsbereich von Namen. Eine Marke gilt in dem Block, in dem sie deklariert wurde, auch in untergeordneten Blöcken, sofern sie nicht dort bereits wieder lokal definiert wurde. Man kann also mit »GOTO« in einen übergeordneten Block springen, nicht aber in einen untergeordneten, da lokale Marken dem übergeordneten Block nicht bekannt sind. Man muß beachten, daß diese Marken nichts mit den Labels in einer »CASE«-Anweisung zu tun haben. Marken und »GOTO«-Sprünge verschlechtern die Übersichtlichkeit eines Programmes erheblich und können mit den in Pascal möglichen Strukturanweisungen auch ohne weiteres vermieden werden. Man sollte sich erst gar nicht angewöhnen, mit Marken zu programmieren.

Damit hätten wir das letzte Kapitel unseres Pascal-Kurses abgeschlossen. Es werden sicherlich Fragen offenbleiben, die hier nicht beantwortet werden konnten. Dies gilt insbesondere für die Kombination und die rekursive Anwendung von Strukturen wie Arrays, Records und vor allem Zeiger. Es sollten auch nur die Grundlagen für die Programmierung in Pascal gegeben werden. Das Verständnis für komplexe Pascal-Strukturen wird sich im Laufe der Zeit selbst einstellen, wenn man sich nur ernsthaft genug mit dieser Programmiersprache befaßt. Sie werden feststellen, daß Pascal eine anwenderfreundliche Sprache ist, deren Komfort Sie bald nicht mehr missen wollen. (Michael Thomas/rf/pf)



Pascal 64 – Nicht nur für Einsteiger

Mit Pascal 64 lohnt sich die Anschaffung der Programmiersprache Pascal auch für den Anfänger. Der Compiler bietet einen großen Sprachumfang und eine komfortable Programmierumgebung. Pascal 64 ist auf Diskette mit Buch erhältlich, das neben dem Compiler auch die Sprache beschreibt.

Pascal findet in den Kreisen der C64-Besitzer immer mehr Anhänger. Für gute Pascal-Compiler mußte allerdings bisher ein stolzer Preis bezahlt werden. Trotzdem steht meist nur ein eingeschränkter Befehlssatz zur Verfügung. Da es auch in Pascal bereits verschiedene Dialekte gibt, ist es für den Einsteiger meist sehr schwierig, sich in diese strukturierte Programmiersprache einzuarbeiten. Mit »Pascal 64« wird ein preisgünstiger Pascal-Compiler angeboten, der mit einem Buch, das sich am Anfänger orientiert, erhältlich ist. Wer sich schon immer mit Pascal befassen wollte, aber noch nicht das richtige Produkt gefunden hat, kann hier mit einer genauen Beschreibung diese »Sprache der tausend Möglichkeiten« kennenlernen.

Große Leistung – kleiner Preis

Eines der wichtigsten Merkmale eines Compilers ist der implementierte Sprachumfang. Daß dieser von Produkt zu Produkt verschieden ist, haben Sie vielleicht beim Vergleich verschiedener Pascal-Compiler bereits festgestellt. Von der Menge der Funktionen und Prozeduren her, die Pascal 64 versteht, kann sich der Compiler durchaus sehen lassen. Zwar wird der Pascal-Kenner die eine oder andere Funktion vermissen, für den Einsteiger reicht der Sprachschatz voll aus. Zu viele Befehle wirken am Anfang eher verwirrend. Nun aber zu den einzelnen Fähigkeiten von Pascal 64. Sehr positiv schlägt der fehlende Kopierschutz zu Buche. Im Handbuch wird sogar auf den ersten Seiten eine genaue Erklärung zum Erstellen einer Arbeitsdiskette gegeben. Dabei wird der Compiler auf die zweite Diskette kopiert. Dafür wird kein spezielles Kopierprogramm benötigt, sondern nur die Basic-Befehle »LOAD« und »SAVE«. Die Lieferdiskette beinhaltet zusätzlich noch einige Beispielprogramme, die zum Teil im Buch besprochen werden. Diese Beispiele behandeln vor allem Compiler-Eigenheiten. Doch zunächst zur Bedienung des Pascal-Compilers. Wollen Sie ein Quellprogramm editieren, kontrolliert das Programm als erstes, ob es nicht schon auf Diskette vorhanden ist, ansonsten wird eine neue Datei angelegt.

Editor für Profis

Der Editor, der sich dann präsentiert, überrascht durch seine Vielzahl an verschiedenen Möglichkeiten (Bild 1). Der versierte Turbo-Pascal-Programmierer erinnert sich hier an die Fähigkeiten dieses Standard-Compilers. So gibt es Funktionen zum Verschieben, Kopieren oder Löschen von vorher markierten Blöcken. Daß die einzelne Zeile über 80 Zeichen gehen kann, ist schon fast selbstverständlich. Dies wird über horizontales Bildschirm-Scrolling erreicht. Der Editor arbeitet mit einer Kommandozeile, die als »TOP« bezeichnet wird. Dort werden alle Anweisungen eingegeben. Nachdem diese mit <SHIFT> und <RETURN> abgeschlossen sind, wird die entsprechende Aktion durchgeführt. Um den Komfort zu vervollständigen, sind im Editor Funktionen zum Suchen und Ersetzen von Strings eingebaut. So können versehentlich und konsequent falsch eingetippte Variablennamen sehr schnell gefunden und durch die richtige Definition ersetzt werden. Erfahrene Programmierer wissen diese Funktion zu schätzen.

Über die Funktionstasten kann das Textfenster, in dem der Quellcode erfaßt wird, beliebig nach oben, unten, links und rechts gescrollt werden. Zwei Funktionstasten dienen zur Fortsetzung eines mit Erfolg abgebrochenen FIND- oder CHANGE-Befehls.

Der Quelltext wird in einem Fenster erfaßt. Außerhalb dieses Textfensters stehen am Rand des Bildschirms die jeweiligen Zeilennummern. Diese werden jedoch nicht etwa als Sprungziele verwendet, sondern dienen eigentlich nur zur Dokumentation und helfen dem Programmierer dabei, Fehler, die bei der Compilation auftauchen, schneller zu finden. Beim Editieren eines neuen Textes erscheinen auf dem Bildschirm die Zeilen »TOP« (Textanfang) und »BOTTOM« (Textende). Um Pascal-Text zu erfassen, müssen als erstes ein paar Zeilen

Editorkommandos	
RESET	Statuszeilen ausblenden
PROFILE	Alle Statuszeilen anzeigen
MSKS	Zeile mit Einfügemaske anzeigen
BNDS	Zeile mit Textgrenzen anzeigen
TAB	Tabulatorzeile anzeigen
COLUMNS	Spaltenkennzeichen anzeigen
END	Text speichern und zurück zum Menü
CANCEL	Zurück zum Menü ohne speichern
SAVE	Altes Textfile löschen, neues speichern
TEXT	Text-Modus einschalten
REPEAT	Alle Tasten erhalten Wiederhol-Funktion
LOCATE n	Zeile n im Text anspringen
FIND	Zeichenfolge suchen
CHANGE	Zeichenfolge suchen und ersetzen
INPUT	sequentielle Datei einlesen
OUTPUT	Text als sequentielle Datei speichern
COPY	Kopieren von Textteilen auf Diskette
Zeilenkommandos:	
I(nsert) n	Es werden n Zeilen eingefügt
D(elete)	Zeile wird gelöscht
DD	Block löschen
R(epeat) n	Zeile wird n-mal wiederholt
M(ove)	Zeile wird neu positioniert
MM	Block zum Verschieben markieren
C(opy)	Zeile wird kopiert
CC	Block zum Kopieren markieren
A(fter)	Block wird hinter die Zeile gestellt
B(efore)	Block wird vor die Zeile gestellt
O(verlay)	Zeile wird überschrieben
OO	Block wird überschrieben

Bild 1. Die vielseitigen Editor-Kommandos von Pascal 64

eingefügt werden. Dann aber erweist sich der Editor von Pascal 64 als komfortabler »Full-Screen«-Editor, wie man es vom Basic des C 64 gewohnt ist. Der Editor kann auf zwei Arten verlassen werden. Einmal wird der Text vor dem Verlassen des Editors auf Diskette gespeichert, oder der Ausstieg erfolgt ohne Sicherung. Es ist jedoch ratsam, den Text immer auf Diskette zu sichern, um den Verlust von Daten oder etwaigen Änderungen zu vermeiden.

Compiler mit Extras

Nach dem Verlassen des Editors kehrt das Pascal-System wieder zum Hauptmenü zurück. Wird der im Speicher befindliche Quelltext nun compiliert, erfolgen zunächst einige Abfragen, mit denen die Grundeinstellung des Compilers verändert werden kann. Als erstes fragt der Compiler nach einer Syntax-Überprüfung. Nach dem Beantworten der Frage mit »ja« werden im Listing alle fehlerhaften Zeilen markiert, jedoch wird kein Source-Code erzeugt. Weiterhin kann die Startadresse des umgewandelten Programms verändert werden. Allerdings sollten Sie hier die Einstellung des Compilers akzeptieren, da ansonsten Eingriffe in das System notwendig werden. Den Compiler-Durchlauf kann man jederzeit durch das Drücken der »*«-Taste abbrechen. Natürlich wird dann kein ablauffähiger Code im Speicher erzeugt, so daß nach vielleicht anstehenden Änderungen wiederum neu compiliert werden muß. Das während des Umwandelns erzeugte Listing kann per Angabe beim Compiler-Aufruf auf den Drucker umgelenkt werden. Ohne ein ausgedrucktes Listing ist eine sinnvolle Problemlösung bei der Komplexität, die ein Pascal-Programm erreichen kann, kaum mehr möglich.

Wurde das Compilieren erfolgreich beendet, steht im Speicher des C 64 ein ausführbares Object-Programm, das allerdings nicht im Pascal-System gestartet werden kann. Um das Programm ablaufen zu lassen, müssen Sie das System verlassen. Dann können Sie das Programm ganz normal mit RUN starten. Durch die Eingabe des Befehls LIST können Sie die Basic-Zeile sehen, die vom Compiler erzeugt wird. Diese enthält lediglich einen SYS-Befehl, der auf die Speicherstelle zeigt, an der das umgewandelte Pascal-Programm beginnt. Sollten Sie beim Testen des Programms noch Fehler feststellen, kann durch Eingabe eines »*« und <RETURN> in das Pascal-System zurückgekehrt und der Quellcode verbessert werden. Für die Korrektur von Programmfehlern muß also die Textdatei, die den Quellcode enthält, nicht immer extra nachgeladen werden, sondern ist neben dem Object-Code ständig im Speicher des C 64 vorhanden. Dadurch wird die Fehlersuche (Debugging) wesentlich erleichtert.

Umfangreicher Sprachschatz

Der Sprachumfang von Pascal 64 (Bild 2) umfaßt eine erstaunlich große Untermenge der Funktionen, die auch die für die größeren Systeme angebotenen Pascal-Compiler enthalten. Alle wichtigen Befehle zur Behandlung von Dateien sind voll implementiert. Sehen wir uns aber zunächst die möglichen Variablentypen näher an. Am meisten überrascht hier, daß die Anweisung PACKED mit angegeben werden kann. Dadurch belegen Typen wie CHAR und BOOLEAN, sowie Aufzählungstypen mit weniger als 256 Elementen, nur noch ein Byte Speicherplatz. Wenn sich der Speicherplatz dem Ende entgegenneigt, empfiehlt es sich also, alle Typen dieser Art zu »packen«. Zu den bisherigen Leistungsmerkmalen kommt noch der Datentyp REAL hinzu, der Gleitkommazahlen aufnehmen kann, was bei einem Compiler dieser Preisklasse nicht unbedingt erwartet werden kann. Der RECORD-Typ bietet bei Pascal 64 beinahe alle Vorzüge, die

Variablentypen:			
BOOLEAN	INTEGER		
STRING	CHAR		
REAL	TEXT		
ARRAY			
vordefinierte Konstanten:			
TRUE	FALSE		
MAXINT			
reservierte Worte:			
AND	FILE	NOT	TO
ARRAY	FOR	OF	TYPE
BEGIN	FORWARD	OR	UNTIL
CASE	FUNCTION	PACKED	VAR
CONST	GOTO	PROCEDURE	WHILE
DIV	IF	PROGRAM	WITH
DO	IN	RECORD	
DOWNTO	LABEL	REPEAT	
ELSE	MOD	SET	
END	NIL	THEN	
Standardprozeduren für dynamische Objekte:			
NEW	MARK		
RELEASE			
Standardprozeduren zur Ein-/Ausgabe:			
CLOSE	EOF	STATUS	
EOLN	PUT	GET	
READLN	READ	WRITELN	
WRITE			
Arithmetische Standardprozeduren:			
ORD	CHR	SUCC	
PRED	ODD	ABS	
INT	TAN	POWER	
Sonstige Prozeduren:			
SYS	POKE	PEEK	
ADDU	HALT		

Bild 2. Der Sprachumfang von Pascal 64

von Turbo-Pascal her bekannt sind. So läßt sich die WITH..DO-Anweisung ohne weiteres auf diesen strukturierten Datentyp anwenden. Für die Dateiverwaltung steht zum Verarbeiten von normalen ASCII-Dateien sogar der Dateityp TEXT zur Verfügung. Man vermißt allerdings die Prozedur SEEK zur Bearbeitung von relativen Dateien. Zum Glück existiert aber eine andere Möglichkeit, um relative Dateien zu bearbeiten. Dazu wird in Pascal 64 eine dynamische Datenstruktur erzeugt. Der Record-Zeiger wird hier als echter Pascal-Pointer definiert, das heißt, der Variablen ist kein fester Wert zugewiesen, sondern sie zeigt auf den entsprechenden Datensatz. Betrachten wir aber jetzt weiter den vorhandenen Sprachumfang. Für Schreib-/Lesezugriffe stehen zwei grundsätzlich verschiedene Möglichkeiten zur Verfügung. Sehen wir uns zunächst die dem Turbo-Pascal bekannten Funktionen zum Lesen und Schreiben von Dateien an. Diese lauten auch in Pascal 64 WRITELN und READLN. Dateien können auf zwei Arten geöffnet werden. Zum einen mit der OPEN-Anweisung, die einer Datei eine logische Nummer zuordnet, zum anderen mit den Pascal-Standard-Prozeduren REWRITE und RESET. REWRITE wird zum Erstellen einer neuen Datei, RESET zum Zurückstellen der Datei auf den Anfang verwendet. Geschlossen werden Dateien wie üblich mit CLOSE. Zum Überprüfen der momentanen Schreib-/Leseoperation bietet Pascal 64 die Statusabfragen EOF und STATUS an. STATUS wird hier wie die Basic-Variable ST (siehe C 64-Handbuch) angewendet. Um Textdateien mit variabler Satzlänge lesen zu können, bietet Pascal 64 die Prozeduren PUT und GET an. Diese holen aus einer

Datei Zeichen für Zeichen in den Speicher. Mit der Funktion EOLN kann auf ein eventuell erreichtes Satzendezeichen geprüft werden. Sie sehen, Pascal 64 bietet dem Programmierer alle Möglichkeiten, die Dateiverwaltung seinen Wünschen entsprechend zu steuern.

Arithmetik eingebaut

Für den versierten Mathematiker stellt Pascal 64 einige Grundfunktionen zur Verfügung, wie sie in jedem Pascal-System vorhanden sind. Die Berechnung des Absolutbetrages einer Zahl ist ebenso wie die des Tangens vorgesehen. Über die Anweisung POWER können beliebige Integer- oder Real-Zahlen potenziert werden.

Wer selbst in Pascal noch Einfluß auf das Betriebssystem nehmen möchte, findet in Pascal 64 leistungsfähige Funktionen zur Manipulation. Mit dem SYS-Befehl werden wie gewohnt Maschinenprogramme aufgerufen. Dem in dieser Beziehung verwöhnten Basic-Programmierer bietet Pascal 64 die Funktionen PEEK und POKE, die beide äquivalent zu den Basic-Befehlen arbeiten.

Selbstverständlich »kennt« Pascal 64 auch alle Strukturanweisungen des Standards. So fehlt weder die FOR..NEXT-Schleife noch die WHILE..DO-Struktur und auch die REPEAT-Anweisung ist vorhanden. Der Programmaufbau hält sich ebenfalls strikt an den Standard. Alle Prozeduren müssen vor der Stelle ihres Aufrufs im Programm definiert werden. Prozeduren und Funktionen kann man mehrere Parameter übergeben, dadurch wird deren Austauschbarkeit erhöht. Leider kann Pascal 64 nur acht KByte Text auf einmal verwalten. Sollten Sie größere Programme schreiben, müßen diese in mehrer Module aufgeteilt und mit der INCLUDE-Anweisung beim compilieren in den Quellcode eingebunden werden.

Damit wäre die Sprachbeschreibung von Pascal 64 abgeschlossen. Über das mitgelieferte Buch ist allerdings noch einiges zu sagen.

Für Einsteiger und Profis

Pascal 64 wurde speziell für den Einsteiger entwickelt. Als besonders wertvoll erweist sich die Konfiguration Buch – Diskette, die dem Anwender die Möglichkeit gibt, praxisorientiert zu lernen. Somit kann das erworbene Wissen sofort in die Tat umgesetzt werden. Das Buch selbst gliedert sich in die drei Teile Pascal-Grundlagen, Tips & Tricks und die Compileranleitung. Im Grundlagenteil werden in einfacher Weise die Datentypen und einfachen Funktionen von Pascal 64 erklärt. Je weiter man blättert, desto tiefer stößt man in die Materie vor. Der Pascal-Schüler wird schrittweise an die Programmierung herangeführt. Beginnend mit ersten kleinen Beispielprogrammen lernt der Anwender immer komplexere Programme kennen und verstehen, so daß einer Nutzung der Sprache nichts mehr im Wege steht. Besonders schwerwiegende Probleme wurden vom Autor des Buches gut erkannt und anhand von ausführlich dokumentierten Programmbeispielen gelöst. Dem Editor von Pascal 64 ist ein eigener Abschnitt gewidmet, der alle Funktionen genau beschreibt.

In der Rubrik Tips & Tricks findet der Anwender und geübte Programmierer nützliche Pascal-Routinen, wie etwa einen Sortier-Algorithmus. Auch zum Editor werden wichtige Hinweise aufgeführt.

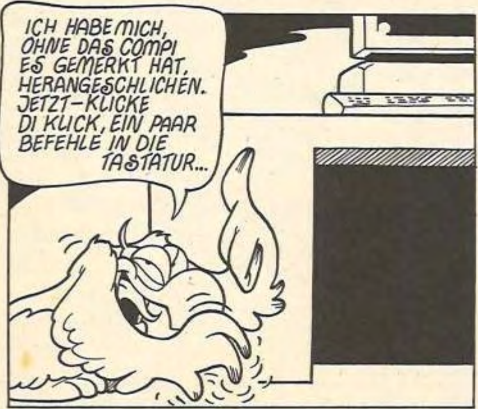
Pascal total

Die Dokumentation des Pascal-Systems schließlich enthält noch einmal in zusammengefaßter Form einen Überblick über die vorhandenen Prozeduren und Funktionen. Am Ende der Einführung findet sich ein Programm, in dem der relative Dateizugriff demonstriert wird. Dieses ist, wie alle anderen Beispiele auch, bestens dokumentiert, so daß der Leser die Gedanken des Programmierers ohne weiteres nachvollziehen kann. Dieser Teil ist vor allem für den Profi gedacht und dient diesem als wertvolles Nachschlagewerk. Des weiteren ist hier die Beschreibung der möglichen Editor- und Compilerfehler untergebracht, die das Interpretieren von möglicherweise auftretenden Fehlermeldungen erleichtert. Schließlich findet sich noch eine Aufzählung aller reservierten Worte in Pascal 64.

Mit Pascal 64 ist es gelungen, ein System anzubieten, das nicht nur leistungsfähig ist, sondern es vor allem dem Einsteiger ermöglicht, sich schnell in Pascal einzuarbeiten. Vor allem der niedrige Preis und die ausführliche Anleitung in Buchform machen dieses Produkt für jeden C 64-Besitzer interessant. Durch den fehlenden Kopierschutz hebt sich der Compiler positiv vom Großteil der C 64-Software ab. Da sich Pascal 64 größtenteils am Standard orientiert, fällt später ohne großes Umlernen der Umstieg auf andere Compiler, wie zum Beispiel Turbo-Pascal, leicht. Somit kann Pascal 64 jedem C 64-Anwender und Pascal-Einsteiger wärmstens empfohlen werden. (rf)

Florian Matthes, »Pascal mit dem C 64«, Markt & Technik Verlag AG, ISBN 3-89090-222-7, Preis: 52 Mark



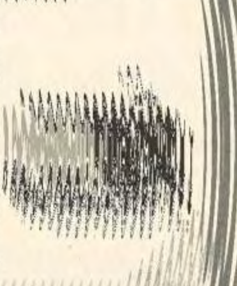


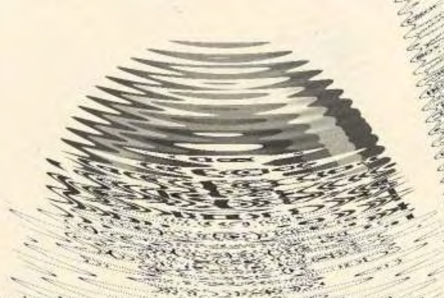


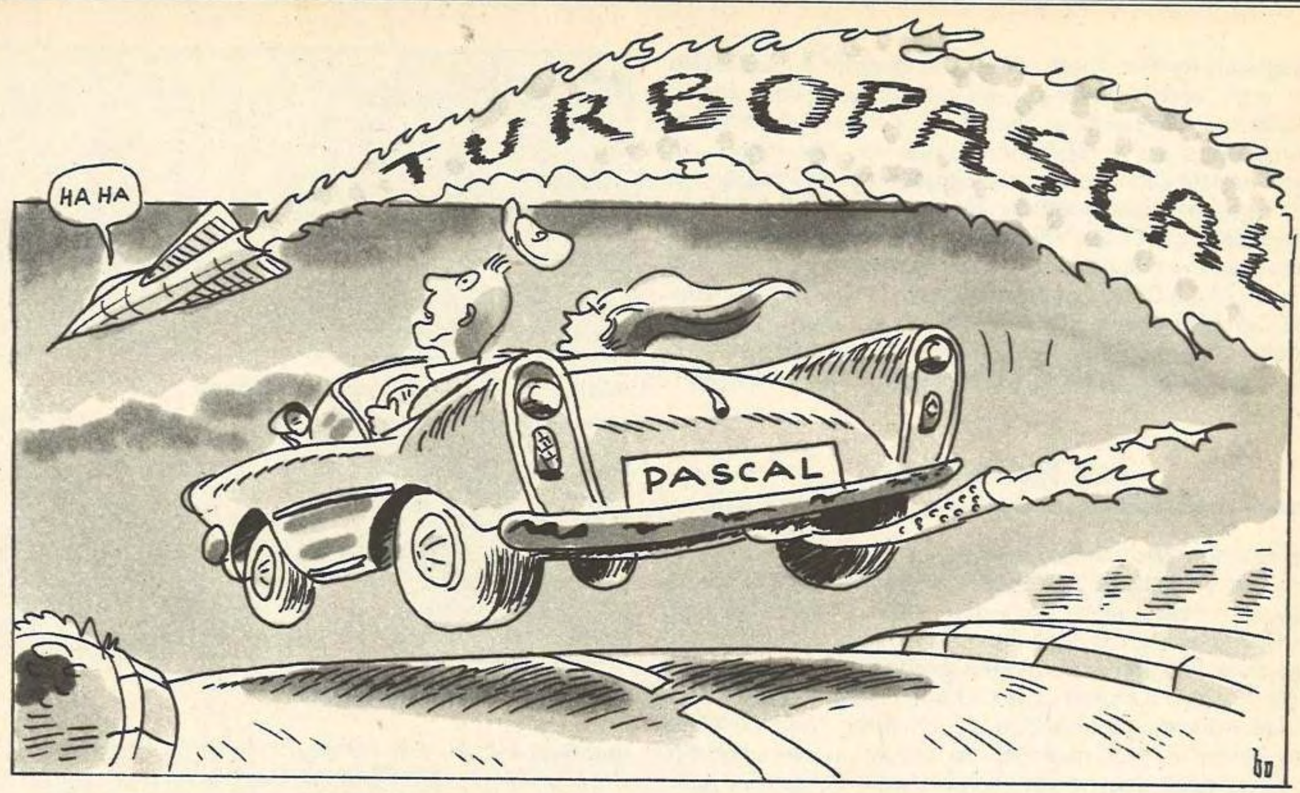





PROTEXT







Schneller, umfangreicher, komfortabler

Seit CP/M auf Heimcomputern wie dem C128 zur Verfügung steht, hat der Pascal-Programmierer Zugriff auf den vielleicht leistungsfähigsten Compiler der Pascal-Welt: Turbo Pascal.

Pascal an sich bedeutete bei seiner Einführung schon eine Revolution bei den Programmiersprachen. Durch die volle Unterstützung der strukturierten Programmierung entwickelte sich Pascal schnell zum Standard in Sachen PC-Software. Einigen Leuten genügte die Leistung der vorhandenen Compiler allerdings nicht ganz. So schufen sie Turbo Pascal. Dieses Software-Produkt besticht nicht nur durch Geschwindigkeit und komfortable Bedienung, sondern auch durch das ungewöhnlich hohe Niveau der Programmierumgebung. Turbo Pascal besteht nicht nur aus einem reinen Compiler, sondern hat zusätzlich einen Editor mit eingebaut, der keine Wünsche offen läßt. Damit sind die Vorteile von Turbo Pascal allerdings noch längst nicht aufgezählt. Wie groß der Ideenreichtum der Entwickler war, erfahren Sie im weiteren. Hier wird nicht auf die Sprache Pascal eingegangen, sondern speziell auf die Zusatzfunktionen von Turbo Pascal. Näheres zur Programmierung in Pascal finden Sie im Pascal-Kurs in diesem Sonderheft.

Editor für gehobene Ansprüche

Voll an Wordstar orientiert sich der Editor von Turbo Pascal. Alle Kommandos dieses leistungsfähigen Textverarbeitungsprogramms stehen auch unter Turbo zur Verfügung. Gegenüber Wordstar weist der Editor sogar noch einige Erweiterungen auf. So wurde er speziell für die Programmeingabe konzipiert. Die Verwendung ist denkbar einfach. Haben Sie den Namen einer Arbeitsdatei definiert, drücken Sie einfach <E>. Das Menü verschwindet und der Editor wird aktiviert. Befindet sich die Arbeitsdatei auf der Diskette, wird sie geladen und die erste Seite am Bildschirm angezeigt. Handelt es sich um eine neue Datei, ist der Bildschirm mit Ausnahme der Statuszeile am oberen Rand leer. Diese Zeile gibt Auskunft über Zeilen- und Spaltennummer, Schreibmodus, Tabulator und die momentan editierte Datei. Insgesamt kennt der Turbo-Editor 45 Befehle, mit denen man durch den Text blättert, Zeichenketten sucht und ersetzt und vieles mehr. Die Editorbefehle lassen sich in drei Kategorien einteilen:
1. Cursor-Steuerbefehle
2. Einfüge- und Löschbefehle
3. Blockbefehle

Alle Befehle werden über die Control-Taste aktiviert. Erwähnenswert ist die Vielfalt der Blockbefehle, sowie die Anweisungen zum Suchen und Ersetzen. Die Blockbefehle sind erweiterte Befehle, das heißt sie bestehen aus zwei Zeichen. Ein Textblock kann nur ein Zeichen, aber auch mehrere Seiten lang sein. Ist der Block markiert, kann er kopiert, bewegt, gelöscht oder in eine Datei geschrieben werden. Mit einem weiteren Befehl kann eine Datei als ein Block in einen Text eingefügt werden. Ein anderer dient dazu, einzelne Wörter als Block zu markieren. Mit dem Suchbefehl können Sie den Text in beliebiger Richtung nach einem maximal 30 Zeichen langen String durchsuchen. Der Such- und Ersetzbefehl sucht nach einer beliebigen Zeichenkette und ersetzt diese durch eine vorher genau definierte. Auch hier beträgt die maximale Länge wieder 30 Zeichen. Beim Editieren fällt noch ein anderer Punkt positiv auf. Sicher haben Sie einmal eines dieser wunderbar strukturierten Pascal-Listings gesehen, bei denen die Strukturierung durch Texteinrückungen noch verstärkt hervorgehoben wird. Turbo Pascal erledigt das

automatisch für Sie. Eine neue Zeile beginnt immer direkt unter dem ersten Zeichen der vorhergehenden. Durch Drücken der <TAB>-Taste rückt der Cursor an das Ende des ersten Wortes der vorstehenden Zeile vor. Sie sehen, ein Programmierer kann sich für die Eingabe seiner Programme nichts Besseres wünschen als diesen ausgereiften und sehr komfortablen Editor. Sie können ihn sogar für die Erstellung von Programmen für andere Compiler verwenden, da ein Text in reinem ASCII-Code gespeichert wird. Da bei den meisten CP/M-Compilern kein Editor enthalten ist, muß man, falls Wordstar oder eben Turbo Pascal nicht zur Verfügung steht, auf den ED zurückgreifen. Da bietet sich Turbo Pascal geradezu an.

Das Turbo-Menü

Unterstrichen wird die Leistungsfähigkeit von Turbo Pascal durch die vielfältigen Menüpunkte (Bild 1). Diese Menüpunkte, zu denen auch der Editor zählt, werden mit dem Anfangsbuchstaben aufgerufen und können, jeder für sich, als eigenes Programm betrachtet werden. Als erstes sehen Sie die Angabe »Logged Drive«. Durch Angabe von »L« können Sie während einer »Sitzung« mit Turbo (so wollen wir Turbo Pascal ab jetzt nennen) das aktive Laufwerk ändern. Turbo fragt nach dem neuen Laufwerksnamen, den Sie dann ohne Doppelpunkt eingeben können.

Der nächste Punkt nennt sich »Work File«. Die Arbeitsdatei (Work File) befindet sich immer im Speicher des Computers. Alle weiteren Befehle wie Edit, Compile und Run werden auf sie angewendet. Mit diesem Punkt können Sie eine neue Datei erstellen oder eine schon existierende in den Arbeitsspeicher laden. Nach dem Eintippen von »W« werden Sie nach dem Namen der Datei gefragt. Der einzugebende Name muß den Konventionen des Betriebssystems entsprechen, also acht gültige Zeichen als Dateinamen und drei als Dateikennzeichnung haben. Wird letztere weggelassen, wird automatisch der Typ ».PAS« angehängt. Noch eine weitere Datei kann in Turbo bearbeitet werden. Diese wird als »Main File« bezeichnet. Sie können nämlich mit der Steueranweisung (diese wird später noch ausführlich besprochen) »(*$I Dateiname.Typ*)« aus einer Datei andere im Quellcode abgelegte Dateien beim Compiliervorgang dazuhängen. Das Programm, das die INCLUDE-Anweisung enthält, muß als Hauptdatei (Main File) definiert werden. Der Menüpunkt wird mit »M« aktiviert. Die Eingabekonventionen entsprechen denen der Arbeitsdatei. Mit zum wichtigsten gehört der Menüpunkt »Compile«. Mit »c« aktivieren Sie den Compiliervorgang. Ist keine Hauptdatei angegeben, wird die Workdatei bearbeitet. Wurde keine von beiden vorher benannt, wird wieder nach dem Namen gefragt. Wird während des Compilierens ein Fehler festgestellt, springt Turbo zurück in den Editor; dabei bleibt der Cursor an der Stelle stehen, an der der Fehler auftrat. Diese Eigenschaft läßt fast vergessen, daß Turbo ein Compiler ist. Das compilierte Programm wird in der Regel im Speicher abgelegt. Wie ein ablauffähiges »COM«-File erzeugt wird, erfahren Sie später.

Zur Fehlersuche (Debugging) bietet Turbo die Auswahl »Run«. Wie üblich wird damit ein Programm gestartet. Befindet sich kein Programm im Speicher, muß erst der Name eingegeben werden. Ein compiliertes Programm wird sofort ausgeführt, andernfalls durchläuft Turbo zuvor automatisch den Punkt »Compile«. Befindet sich das Programm, beispielsweise durch einen Programmierfehler, in einer Endlosschleife, kann es durch die bei CP/M übliche Tastenkombination <CTRL+C> abgebrochen werden. Allerdings muß dazu wieder eine bestimmte Compiler-Option gesetzt werden. Mit dem Menüpunkt »Save« schließlich wird die Arbeitsdatei mit dem Kürzel ».PAS« auf Diskette gespeichert. Nun

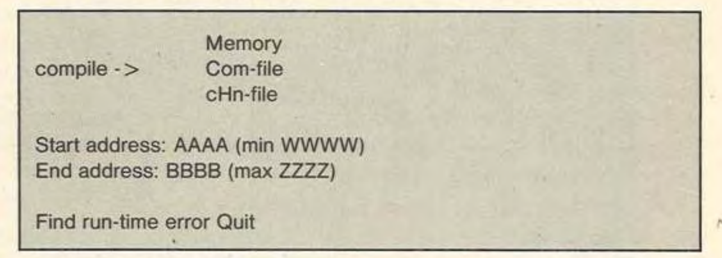
```
Logged drive: LW
Active directory: \NAME

Work file:
Main file:

Edit        Compile     Run     Save
eXecute     Dir         Quit    compiler Options

Text:            0 bytes
Free:        62024 bytes
```

Bild 1. Das Hauptmenü von Turbo Pascal



```
              Memory
compile ->    Com-file
              cHn-file

Start address: AAAA (min WWWW)
End address: BBBB (max ZZZZ)

Find run-time error Quit
```

Bild 2. Untermenü mit Optionen

kommen wir zu einer Auswahl des Turbo-Systems, die dem Programmierer das Tor zum Betriebssystem öffnet: »eXecute«. Sie können damit beliebige, über CP/M lauffähige Programme aufrufen. Nach der Meldung »Program:« tippen Sie den gewünschten Namen ein. Sie können beispielsweise eine Textdatei mit Wordstar bearbeiten, mit PIP Programme kopieren oder mit SHOW den Disketten-Speicherplatz überprüfen. Erst nach dem Programmende kehren Sie wieder nach Turbo zurück. Sie sehen das an der Bereitschaftsanzeige (>). Durch Drücken der <LEER>-Taste wird das Hauptmenü wieder angezeigt. Damit diese Option funktioniert, muß sich auf der Diskette die Datei »TURBO.OVR« befinden.

Die Auswahl »Dir« ist mit dem entsprechenden CP/M-Programm identisch und mit »Quit« kann Turbo verlassen werden.

Compiler mit Optionen

Nach der Eingabe von »O« erscheint ein Untermenü (Bild 2) von Turbo, das je nach der angewählten Compiler-Option anders aussehen kann. Mit dem Text »compile« wird wie mit einem Zeiger auf die aktive Option gewiesen. In der Standardeinstellung (Memory = Speicher) wird der compilierte Objektcode im Speicher abgelegt. Ist diese Einstellung nicht schon gesetzt, kann sie mit »M« aktiviert werden. Natürlich können Sie mit Turbo, wie bereits angesprochen, auch ein ablauffähiges »COM«-File erzeugen. Dazu stellen Sie den Zeiger durch Drücken der Taste <C> auf »Com-file«. Nun haben Sie sogar die Möglichkeit, die Startadresse, sowie die maximale Endadresse zu ändern. Dies ist allerdings nur dann notwendig, wenn die Programme beispielsweise auf einer anderen Computerumgebung mit weniger Speicherplatz laufen oder wenn vor dem eigentlichen Programm absolute Variablen abgelegt werden sollen.

Zu guter Letzt können Sie über die Option »H« noch Chain-Dateien erzeugen. Dabei ist zu beachten, daß solche Programm-Module in eine COM-Datei eingebunden werden müssen und die Startadresse der Chain-Datei mit der des COM-Files übereinstimmen muß.

Sie wissen bereits, daß Turbo, wenn ein Programm mit RUN gestartet wird und abbricht, automatisch zum Editor zurückkehrt. Was aber ist zu tun, wenn außerhalb der Turboumgebung ein COM-File plötzlich mit einem Fehler aussteigt?

Prozeduren		
Append	GetDir	OvrDrive
Assign	GetMem	OvrPath
Bdos	GotoXY	Randomize
Bios	Erase	Read
BlockRead	Execute	Readln
BlockWrite	Exit	Release
Chain	FillChar	Rename
Close	InsLine	Reset
ClrEol	Intr	Rewrite
ClrScr	LowVideo	RmDir
CrtExit	Mark	Seek
Crtlnit	MkDir	Str
DelLine	Move	Truncate
Delay	MsDos	Val
Delete	New	Write
Dispose	NormVideo	Writeln
FreeMem		

Bild 3. Der Sprachumfang von Turbo Pascal

Um den Vorteil der direkten Fehleranzeige genießen zu können, gibt es bei Turbo die Option »Find run-time-error«. Um diese Option richtig einzusetzen, notieren Sie sich den beim Programmausstieg angezeigten Programm-Counter (PC). Rufen Sie Turbo auf und laden Sie mit »W« die Textdatei Ihres fehlerhaften Programms. Wählen Sie die Option »O« und dort den Punkt »F«. Auf die Frage »Enter PC:« geben Sie den notierten Programmzählerstand ein. Der Editor wird aktiviert und der Cursor bleibt auf der fehlerhaften Stelle stehen.

Die Compiler-Direktiven

Turbo Pascal eignet sich also nicht nur zur Programmierung, sondern unterstützt den Programmierer bei der Quellcode-Eingabe ebenso wie beim Debugging. Damit wäre dem Programmierer eigentlich schon weit geholfen, doch Turbo bietet noch mehr.

Eine Anzahl der Eigenschaften des Turbo-Compilers werden durch Compiler-Anweisungen (Direktiven) gesteuert. Eine Compiler-Steueranweisung wird als Kommentar mit besonderer Syntax eingebracht. Das bedeutet, daß immer dann, wenn ein Kommentar in einem Programm erlaubt ist, auch eine Compiler-Steueranweisung stehen darf. Eine solche Steueranweisung besteht aus einer sich öffnenden geschweiften Klammer. Danach folgt unmittelbar ein Dollarzeichen (<$>), an das die Steueranweisung direkt anschließt. Sie besteht grundsätzlich aus einem Buchstaben. Es können auch ganze Listen angegeben werden, die durch Kommas voneinander getrennt werden. Als letztes steht eine sich schließende geschweifte Klammer. Anstelle der geschweiften Klammern können auch die Zeichen »(*« und »*)« stehen. Weder vor noch hinter dem Dollarzeichen dürfen Leerzeichen stehen. Ein Pluszeichen (+) zeigt an, daß die Compiler-Eigenschaft eingeschaltet ist (aktiv), ein Minuszeichen (-) definiert die Eigenschaft als ausgeschaltet (passiv).

Alle Compiler-Steueranweisungen haben Standardeinstellungen. Sie sind so gewählt, daß sie die maximale Ausführungsgeschwindigkeit und die minimale Codegröße einnehmen. Deshalb ist die Codeerzeugung für rekursive Prozeduren und die Index-Überprüfung standardmäßig nicht eingeschaltet. Überprüfen Sie vorher, ob Sie die gewünschten Steueranweisungen gesetzt haben. Im Anschluß folgt nun eine genaue Beschreibung der einzelnen Direktiven.

Die erste Steueranweisung läßt sich durch ein »B« darstellen. Damit wird der I/O-Modus gesteuert. Ist sie aktiv (»(*$B+*)«), ist die Konsole den Standarddateien INPUT und OUTPUT zugewiesen, also den Ein-/Ausgabekanälen. Ist sie passiv (»(*$B-*)«), wird das Terminal verwendet. Diese Steueranweisung gilt für einen gesamten Programmblock und kann innerhalb des Programms nicht neu definiert werden. Diese Anweisung sei hier nur der Übersicht halber aufgeführt, da die Terminalzuweisung für den C128 nicht vorgesehen ist.

Mit der Direktive »C« werden die Steuerzeichen während der Konsolen-Ein-/Ausgabe gesetzt. Ist sie aktiv (»(*$C+*)«), unterbricht <CTRL+C> als Antwort auf eine READ oder READLN-Anweisung die Programmausführung. Mit <CTRL+S> wird die Bildschirm-Ausgabe ein- und ausgeschaltet. Ist sie passiv (»(*$C-*)«), werden Steuerzeichen nicht interpretiert. Der aktive Zustand verlangsamt die Bildschirm-Ausgabe. Wollen Sie also eine hohe Ablaufgeschwindigkeit erreichen, sollten Sie diese Steueranweisungen ausschalten. Die Steueranweisung ist für einen gesamten Programmblock gültig und kann während des Programms nicht neu definiert werden.

Die Direktive »I« weist in Turbo eine Doppelfunktion auf. Die erste Variante wird zur I/O-Fehlerbehandlung verwendet. Ist diese Steueranweisung aktiv (»(*$I+*)«), werden alle Ein-/Ausgabeoperationen automatisch auf Fehler hin überprüft.

Ist sie passiv (»(*$I-*)«), muß der Programmierer selbst mit Hilfe der Standardfunktion »IOResult« möglicherweise aufgetretene Fehler ermitteln. Die zweite Anwendung von »I« bezieht sich auf das nachträgliche Laden von Programm-Modulen. Steht nach der »I«-Anweisung ein Dateiname, wird die Datei mit diesem Namen in das Programm eingebunden und compiliert.

Für die Index-Bereichsüberprüfung steht die »R«-Option zur Verfügung. Diese Steueranweisung kontrolliert die Indexüberprüfung während des Programmlaufs. Ist sie aktiv (»(*$R+*)«), wird bei allen Matrix-Indizierungsoperationen überprüft, ob die Werte innerhalb des definierten Bereichs liegen. Das gilt auch bei allen Zuweisungen für Skalare und Unterbereichsvariable. Ist sie passiv (»(*$R-*)«), wird keine Überprüfung vorgenommen. Dann kann es vorkommen, daß Indexfehler das Programm durcheinanderbringen. Deshalb ist es am besten, wenn Sie diese Steueranweisungen bei der Entwicklung Ihrer Programme verwenden. Haben Sie die Fehler gefunden, wird die Durchlaufzeit durch Ausschalten der Direktive wieder kürzer.

Debugging mit Komfort

Die »V«-Compiler-Anweisung steuert die Typüberprüfung bei Zeichenketten, die in Unterprogrammen als »VAR«-Parameter übergeben werden. Ist sie aktiv (»(*$V+*)«), wird die Überprüfung ausgeführt. Das heißt, daß die Längen der aktuellen und formalen Parameter übereinstimmen müssen. Ist sie passiv (»(*$V-*)«), erlaubt der Compiler auch die Übergabe der aktuellen Parameter, die mit der Länge der formalen Parameter nicht übereinstimmen.

Mit der »U«-Steueranweisung werden die Anwenderunterbrechungen gesteuert. Sobald sie aktiviert wird (»(*$U+*)«), kann der Anwender zu einer beliebigen Zeit während der Ausführung durch Eingabe von <CTRL-C> den Programmlauf unterbrechen. Im anderen Fall gibt es keine Möglichkeit zur Unterbrechung des laufenden Programms. Während der Testphase sollte dieser Schalter auf jeden Fall gesetzt werden. Selbst wenn das Programm in der Turbo-Umgebung getestet wird, besteht keine Möglichkeit, den Ablauf zu unterbrechen. Steckt das Programm in einer Endlosschleife, hilft nur noch das erneute »Booten« des CP/M-Systems.

Die »A«-Direktive hat Einfluß auf die Erzeugung von absolutem, also nicht rekursivem Code. Nach der Aktivierung (»(*$A+*)«) wird absoluter Code erzeugt. Ist dieser Schalter nicht gesetzt, erzeugt der Compiler Code, der rekursive Aufrufe ermöglicht. Dieser Code belegt mehr Speicherplatz und macht die Ausführung langsamer.

Die »W«-Steueranweisung regelt die Ebenen der verschachtelten WITH-Strukturen. Damit wird die Anzahl der RECORDs, die innerhalb eines Blocks geöffnet sind, festgelegt. Nach »W« muß unmittelbar eine Ziffer zwischen eins und neun stehen.

Die letzte Direktive wird zur Matrizenoptimierung eingesetzt und läßt sich durch den Buchstaben »X« ansprechen. Nach der Aktivierung wird die Codeerzeugung für Matrizen auf maximale Geschwindigkeit optimiert. Ist die Anweisung passiv, wird statt dessen die Codegröße minimiert.

Nun haben Sie einen Überblick über die Fähigkeiten des Compilers erhalten. Was jeden Pascal-Programmierer interessiert, ist der in Turbo implementierte Sprachumfang. Auch darüber gibt es nur Positives zu berichten.

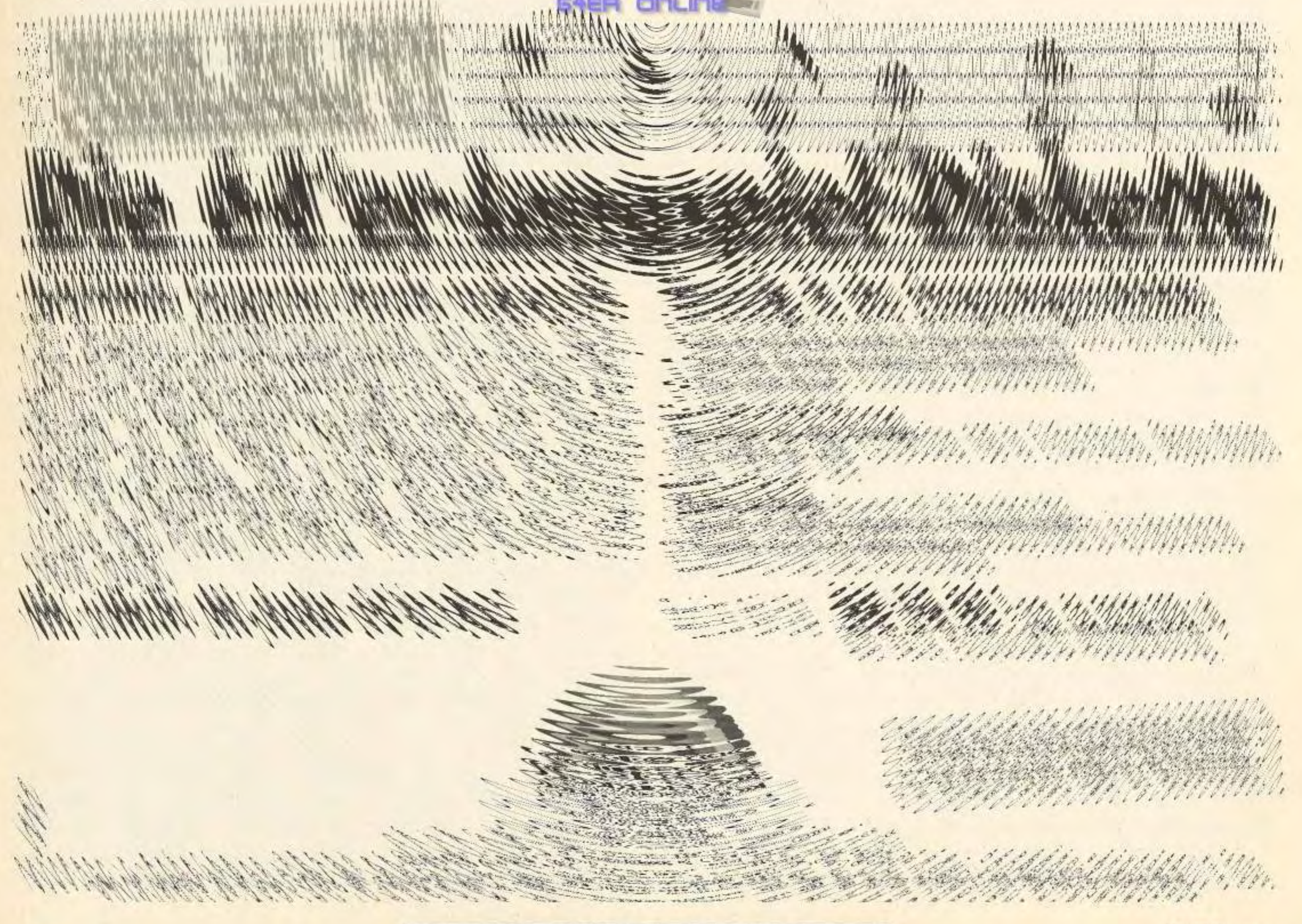
Erweiterter Sprachumfang

Als Standard bei den »normalen« Pascal-Compilern hat sich das USCD-Pascal durchgesetzt. Wir zeigen Ihnen die gravierendsten Unterschiede zwischen USCD- und Turbo-Pascal (Bild 3) auf. Am meisten fallen die Zusatzfunktionen von Turbo im Bereich der Dateiverwaltung auf. Während in den meisten herkömmlichen Pascal-Versionen mit Datenpuffern gearbeitet wird, arbeitet Turbo Pascal mit vordefinierten Prozeduren, die die Handhabung von Dateien erleichtern. Damit aber noch nicht genug der Unterschiede. In Turbo sind bereits eine Menge Standardoperatoren vorhanden. Diese dienen zum Abfragen bestimmter Zustände während des Programmablaufs. So können beispielsweise jederzeit Ein-/Ausgabefehler über die Funktion IOResult abgefragt werden, wenn die entsprechende Compiler-Option eingestellt ist (siehe weiter oben). Für die Bildschirmsteuerung stehen dem Pascal-Programmierer ebenfalls viele Prozeduren zur Verfügung. So ist es kein Problem, einzelne Zeilen zu löschen oder den Cursor auf einen bestimmten Bildschirmpunkt zu positionieren. Wenn man länger mit Turbo Pascal gearbeitet hat, möchte man den Komfort und die Erweiterungen, die das System bietet, auf keinen Fall mehr missen.

Lohnt sich Turbo Pascal?

Diese Frage kann guten Gewissens mit »ja« beantwortet werden. Schon wegen der Benutzerfreundlichkeit des gesamten Programmpakets, wie auch der Popularität der Sprache, kann Turbo Pascal jedem CP/M-Anwender empfohlen werden. Von Systemen vergleichbarer Leistung ist man in der Zwischenzeit stolze Preise gewohnt. Turbo Pascal hingegen kostet für den C 128 nur zirka 219 bis 250 Mark, je nach Anbieter. Dieses Angebot hält einem Preis-/Leistungsvergleich in jedem Fall stand. Die Unterschiede zum Pascal-Standard stellen dabei keine Einschränkung, sondern eher eine Erweiterung dar. Turbo Pascal, leicht erlernbar und kostengünstig, erweist sich als eines der besten Programm-Entwicklungssysteme, die für CP/M momentan erhältlich sind. (rf)

Info: Heimsoeth Software, Fraunhoferstraße 13, 8000 München 5



Strukturiertes Programmieren in Comal

Comal ist eine Programmiersprache mit erstaunlichen Fähigkeiten. Da sie die besten Eigenschaften vieler bekannter Sprachen enthält, ist sie zudem leistungsfähig und benutzerfreundlich.

Vor zwei Jahren nur einem engen Kreis von Spezialisten bekannt, macht auch in der Bundesrepublik Deutschland eine neue Computersprache Furore: Comal.

1973 wurde sie in Dänemark von Borge Christensen mit dem Ziel entwickelt, eine Alternative zu dem unstrukturierten Basic und dem schwer zu handhabenden Pascal für den Einsatz an allen Schulen, aber auch für anspruchsvollere Anwendungen zu schaffen. Die ersten Versuche in dieser Richtung bedeuteten nicht mehr als eine Basic-Erweiterung (wie zum Beispiel Simons Basic). Doch sehr bald löste man sich von den alten Grenzen. Herr Christensen entwarf ein Konzept, das sich als richtungweisend herausstellen sollte.

Das Rezept war schnell formuliert: Die neue Sprache (in Zukunft Comal 80) genannt, sollte folgende Bedingungen erfüllen:

1) Strukturiertheit wie in Pascal
2) direkte Interaktivität wie in Basic
3) anfängergerechte Handhabung der grafischen Fähigkeiten eines Computers wie bei Logo
4) Übertragbarkeit von Programmen auf möglichst viele Computer

Nach kurzer Zeit lagen die ersten Implementationen vor. Die wichtigste von diesen nannte sich später COMAL 0.14 und lief auf dem Commodore 64. Aufgrund von Differenzen zwischen den Entwicklern und der damaligen Auftragsfirma wurde diese Version allgemein freigegeben: Sie kann heute von jedermann kopiert und weitergegeben werden!

Dies trug natürlich enorm zur Verbreitung der Sprache bei. Die skandinavischen Länder, Irland, Schottland, Kanada, USA... Überall wurde Comal mit Begeisterung aufgenommen. Seit sich in Deutschland eine Gruppe um die Verbreitung und Pflege der Sprache bemüht, fallen auch hier immer mehr Schranken. In vielen Bundesländern gilt COMAL inzwischen als die einzige Sprache, welche zugleich die meisten Anforderungen des Lehrplanes erfüllt und andererseits sehr schnell und ohne Mühe erlernt werden kann. Seit kurzem reagieren auch die Schulbuch- und Zeitschriftenverlage darauf.

Was ist nun an dieser Sprache besonderes, daß sie solche Erfolge hat? Wir werden dies etwas genauer untersuchen.

Comal hat lange Namen

Vorbei sind die Zeiten der kurzen und daher unverständlichen Variablennamen. Wenn man sagen kann

Durchschnitt:=Summe_Werte/Anzahl_Werte

so ist hier schon durch die Formulierung eine Klarheit und Eindeutigkeit vorgegeben, welche auch nach Monaten erhalten bleibt. Versuchen Sie einmal, nach einem halben Jahr in einem Basic-Programm die Bedeutung der verwendeten Variablen zu enträtseln! Selbst in dem Programm, an welchem Sie gerade arbeiten, haben Sie oft genug Schwierigkeiten: Was bedeutete eigentlich wieder »fu$«?

Namen dürfen in Comal maximal 63 Zeichen haben, was für alle Fälle ausreichend ist. Dazu kommt, daß für die Namen der Label (= Sprungmarken), Prozeduren (eine Form von Unterprogrammen) und Funktionen dasselbe gilt.

Comal erleichtert das Umsteigen

Nahezu jeder hat schon einmal mit Basic gearbeitet und kennt dessen Befehlssatz. Deshalb hat man sich entschlossen, in Comal einen Standard von Basic her zu übernehmen. Andererseits wurden jedoch diese Befehle mit erweiterten Möglichkeiten ausgestattet. So können Sie eine FOR-NEXT-Schleife wie in Basic eingeben:

```
FOR i=1 TO 10
PRINT i
NEXT
```

Sobald Sie jedoch diese Zeilen LISTen, schauen sie etwas anders aus, da Comal sie sofort in seine eigene Schreibweise umsetzt:

```
FOR i:=1 TO 10 DO
   PRINT i
ENDFOR i
```

Doch beschränkt sich Comal nicht auf die Nachahmung von Basic. Es bietet erweiterte Möglichkeiten an:

```
FOR i:=1 TO 10 DO PRINT i
```

Diese Zeile ist eine Kurzform der FOR-NEXT-Schleife.

Comal ist strukturiert

Sehr bald hat man erkannt, daß eine rationelle Programmentwicklung ohne bestimmte Regeln nicht möglich ist. Eine der wichtigsten dieser Regeln besagt, daß die Struktur eines Programmes möglichst sofort erkannt werden soll, ja daß ein Programm mit Hilfe geeigneter Strukturen zuerst auf dem Papier entwickelt wird, bevor es zur Umsetzung im Computer, der eigentlichen Codierung in der gewählten Sprache, kommt.

In Comal wurde diese Regel derart berücksichtigt, daß es sogar schwerfällt, unstrukturiert ein Programm zu schreiben. Dies geht soweit, daß einfache Hierarchien, wie zum Beispiel Schleifen, schon beim Auslisten eines Programmes dadurch gekennzeichnet werden, daß die Strukturkörper eingerückt dargestellt sind. Sie haben dies weiter oben schon am Beispiel der FOR-Schleife gesehen, man kann es aber noch deutlicher aufzeigen:

```
FOR schleifenzähler:=1 TO endwert DO
  PRINT schleifenzähler
  FOR zweite_schleife:=1 to 5 DO
    PRINT "innere Schleife"
  ENDFOR zweite_schleife
ENDFOR schleifenzähler
```

Auf Anhieb erkennt man, welche Anweisungen in welche Struktur gehören: Alle Befehle der gleichen Hierarchie sind gleichweit eingerückt.

Doch nicht nur Äußeres macht die Strukturierung aus. Comal bietet mehr. Dies beginnt bei den einfachen Schleifen und Verzweigungen und endet mit dem Konzept der Prozeduren und Funktionen.

Neben der schon angesprochenen FOR-ENDFOR-Schleife gibt es noch weitere Formen:

```
WHILE bedingung DO
        block
ENDWHILE
REPEAT
        block
UNTIL bedingung
LOOP                        (nicht in 0.14!)
        block
        EXIT WHEN bedingung
        ...
        EXIT
        ...
ENDLOOP
```

Sie sehen, Comal ist reich gesegnet mit Möglichkeiten der Gestaltung von Schleifen. Wem dies zuviel oder verwirrend erscheint, dem sei gesagt: Gerade durch die vielen Möglichkeiten können die meisten Algorithmen genauso in Comal geschrieben werden, wie sie in der Umgangssprache formuliert wurden:

Umgangssprache:
SOLANGE die Länge eines Textes kleiner als 20 Zeichen ist, hänge ein Leerzeichen am Ende an.

Comal:

```
WHILE LEN(text$)<20 DO
  text$:+" "
ENDWHILE

(WHILE LEN(text$)<20 DO text$:+" ")
```

Umgangssprache:
Wiederhole:
  Verdopple die Zahl
Wenn die Zahl ohne Rest durch 4 teilbar ist, dann gebe sie aus bis die Zahl größer als 100 ist.

Comal:

```
REPEAT
        zahl:=Zahl*2
        IF (zahl MOD 4)=0 THEN
                PRINT zahl
        ENDIF
UNTIL zahl>100
```

Umgangssprache:
Lege die Datenformate fest.
Schleife für immer:
lösche den Bildschirm
schreibe den Menütext
hole Tastendruck
untersuche den Tastendruck:
wenn die Taste »0« gedrückt ist, beende das Programm
wenn die Taste »1« gedrückt ist, gehe zur Dateneingabe
wenn die Taste »2« gedrückt ist, ändere Daten
wenn die Taste »3« gedrückt ist, suche Daten
wenn die Taste »4« gedrückt ist, gebe die Daten aus
sonst gebe Fehlermeldung
Ende der Untersuchung
Ende der Schleife
Comal:

```
Datenformate_festlegen
LOOP
  PAGE
  menuetext_schreiben
  REPEAT
    taste$:=KEY$
    UNTIL taste$ IN "0123456789"
  CASE taste$ OF
  WHEN "0"
    EXIT
  WHEN "1"
    neue_Daten_eingeben
  WHEN "2"
    Daten_aendern
  WHEN "3"
    Daten_suchen
  WHEN "4"
    Daten_ausgeben
  OTHERWISE
    PRINT "Eingabefehler!"
    PRINT "weiter mit Taste.."
    WHILE KEY$="" DO NULL
  ENDCASE
ENDLOOP
```

Gerade am letzten Beispiel sehen Sie, wie schnell ein Programm aufgebaut werden kann. Sie brauchen hier nur noch die entsprechenden Prozeduren zu schreiben, dann ist das Programm fertig. Zum Austesten genügt es, einfach leere Prozeduren zu schreiben. Was Prozeduren sind, erfahren Sie weiter unten.

Gleichzeitig führt das letzte Beispiel zu den weiteren Strukturierungsmöglichkeiten hin: den bedingten Verzweigungen.

Comal kennt viele Möglichkeiten, anhand einer Bedingung im Programm verschiedene Wege zu gehen. Die einfachste davon ist wieder aus Basic übernommen:

```
IF zahl1>zahl2 THEN DO aktion
```

Oft müssen bei einer erfüllten Bedingung jedoch mehrere Aktionen erfolgen. Jede Folge von Befehlen nennt man einen Block. In Comal gilt die Regel: Überall dort, wo ein Befehl stehen darf, darf auch ein Block stehen. Die Ausnahme bilden die Kurzformen von Strukturen. Doch auch hier kann man eben einen Namen schreiben, welcher beide Erfordernisse zugleich erfüllt: Er ist nur ein Befehl, aber sein Aufruf bewirkt die Abarbeitung eines kompletten Blocks.

Verwendet man mehrere Befehle, so lautet die obige Struktur:

```
IF bedingung THEN
  ...block...
ENDIF
```

Zwischen IF und ENDIF darf beliebig viel stehen, genau wie zum Beispiel in Pascal. Überhaupt werden Sie immer wieder eine starke Ähnlichkeit mit dieser Sprache feststellen können.

Was tun wir, wenn wir, abhängig von der jeweiligen Bedingung, unterschiedliche Befehle ausführen lassen wollen? Comal hilft auch hier weiter:

```
IF bedingung THEN
  Block für "Bedingung wahr"
ELSE
  Block für "Bedingung falsch"
ENDIF
```

Nehmen wir nun folgendes Problem an: Sie wollen Ihren Abend verplanen. Dabei sagen Sie: Wenn meine Freundin kommt, bleibe ich zu Hause. Wenn es regnet, gehe ich in ein Konzert. Andernfalls möchte ich einen Spaziergang machen. In Comal würde dies so aussehen:

```
IF freundin_kommt THEN
  zu_hause_bleiben
ELIF regen=TRUE   (TRUE heist WAHR)
  gehe_ins_Konzert
ELSE
  Spaziegang
ENDIF
```

In Pascal wäre dies nur durch eine Schachtelung mit IF..ELSE IF..ENDIF..ENDIF erreichbar, in Basic müßten Sie verwirrend im Programm springen.

Als wir am Anfang ein kleines Programm geschrieben haben, haben wir die mächtigste Form der Verzweigung kennengelernt: die CASE-Struktur. Hier wird ein Ausdruck, meist eine Variable, als Kontrolle benutzt, anhand dessen die einzelnen Möglichkeiten mit WHEN direkt angegeben und entsprechende Aktionen ausgeführt werden. Tritt ein Fall ein, welcher nicht durch ein passendes WHEN abgedeckt ist, so wird in den OTHERWISE-Teil gesprungen.

Comal ist schnell

Bei der Konstruktion von Comal mußte ein Kompromiß geschlossen werden: Schnelligkeit und Benutzerfreundlichkeit vertragen sich schlecht. Aber es ließ sich doch machen:
- Die Rechengeschwindigkeit wurde durch bessere Routinen erhöht.
- Die Zeichenverarbeitung geschieht extrem schnell, da nicht mit der dynamischen Speicherverwaltung von Basic gearbeitet wird. Deshalb entfällt auch die gefürchtete Garbage Collection. Comal arbeitet bis zu 70mal schneller als das Commodore-Basic, wobei Basic-Erweiterungen wie Simons Basic übrigens noch langsamer sind.
- Es wurde eine interne Darstellung des Programmes gewählt, welche die Ausführungszeiten optimiert. Dabei wurde die Konzeption der Sprache Forth teilweise übernommen.
- Schon während der Eingabe findet eine erste Übersetzung statt, welche durch eine weitere Phase nach RUN ergänzt wird. So wird beispielsweise bei ENDLOOP gleich die Speicheradresse des dazugehörigen LOOP mit abgelegt, und es braucht während des Programmablaufes kein Ziel mehr errechnet zu werden.
- Da ebenfalls schon bei der Eingabe die Syntax überprüft wird, kann diese bei der Ausführung entfallen. Natürlich müssen Laufzeitfehler trotzdem berücksichtigt werden, so zum Beispiel ein Zahlenüberlauf oder der Aufruf einer nicht vorhandenen Prozedur.

Comal unterstützt die Grafik

Ein großes Manko des im C 64 eingebauten Basic ist das vollständige Fehlen von Grafikbefehlen. Dem wurde Rechnung getragen, wobei man ein bereits bewährtes und eingeführtes Konzept zugrunde legte: Das Zeichnen mit der »Turtle«, einer Dreiecksfigur, welche auch im Direktmodus angesprochen werden kann. Dies macht die Sprache Logo für Einsteiger so interessant.

Die Befehle sind einfach und verständlich. Wer das Ganze in deutsch haben möchte, kann dies sehr einfach ohne gewichtigen Geschwindigkeitsverlust tun. Ein Beispiel:

```
PROC vorwärts(schritte)
  forward(schritte)
ENDPROC vorwärts
PROC rechts(winkel)
  right(winkel)
ENDPROC rechts
```

Dabei ist man so klug gewesen, bei den Winkelangaben unser gewohntes System beizubehalten: Man arbeitet mit Grad-Angaben. Dies erleichtert manches. So kann durch folgende Befehle sehr schnell ein Rechteck gezeichnet werden:

```
vorwärts(höhe)
rechts(90)
vorwärts(breite)
rechts(90)
vorwärts(höhe)
rechts(90)
vorwärts(breite)
rechts(90)
```

Man programmiert genauso, als ob man mit einem Bleistift auf einem Blatt Papier zeichnen würde.

Prozeduren & Funktionen

Das weitaus mächtigste Sprachelement in Comal ist das Konzept der Prozeduren und Funktionen. Darunter hat man sich eine Art von Unterprogrammen vorzustellen. In Comal sind diese jedoch mit einigen Besonderheiten ausgestattet.

Zum einen werden Funktionen und Prozeduren anhand ihres Namens aufgerufen. Dies korrespondiert mit dem Prinzip, keine Bezüge auf Zeilennummern zuzulassen.

Zum anderen können sogenannte Parameter übergeben werden. Dabei gibt es prinzipiell zwei Typen:

1. Der Werteparameter ist nichts anderes als eine Kopie eines Wertes. Übergeben Sie zum Beispiel eine Variable, so wird deren Inhalt benutzt. Am Ende ist aber wieder der alte Inhalt vorhanden. Man nennt dies auch Lokalität.

2. Der Variablenparameter ist gekennzeichnet durch das Wort REF. Eine Veränderung an diesem Parameter ist endgültig und wird dann auch dem aufrufenden Programmteil bekannt.

Bei Funktionen gilt es schließlich noch zu bedenken, daß unbedingt etwas zurückgegeben wird. Dies kann eine ganze Zahl, eine Fließkommazahl oder ein Text sein. Entsprechend wird auch der Typ der Funktion definiert.

Allgemein sind Prozeduren und Funktionen so aufgebaut:

```
PROC name[(Parameterliste)] [CLOSED]
   ...block..
ENDPROC name

FUNC name[$/#][(Parameterl.)][CLOSED]
   ...block...
   RETURN funktionsergebnis
   .....
ENDFUNC name
```

Wir wollen das verdeutlichen. Unsere Befehlsfolge zum Zeichnen eines Rechteckes soll mit einem Namen versehen werden:

```
PROC rechteck(höhe,breite)
 vorwärts(höhe)
 rechts(90)
 vorwärts(breite)
 rechts(90)
 vorwärts(höhe)
 rechts(90)
 vorwärts(breite)
 rechts(90)
ENDPROC rechteck
```

Nachdem Sie einmal RUN gesagt haben, verhält sich diese Prozedur wie eine Befehlserweiterung. Wenn Sie im Direktmodus eingeben

```
rechteck(10,50)
```

so wird auch das entsprechende Rechteck gezeichnet.

```
FUNC durchschnitt(wert(),anzahl) CLOSED
 summe:=0
 FOR i:=1 TO anzahl DO
  summe:+wert(i)
 ENDFOR i
 RETURN summe/anzahl
ENDFUNC durchschnitt
```

Wie Sie sehen, kann ein komplettes Feld als Parameter übergeben werden. Aufgerufen wird diese Funktion wie eine bereits eingebaute, zum Beispiel »SIN« oder »LEN«.

Durch das Wort CLOSED gilt die Funktion absolut als lokal. Außerhalb verwendete Namen sind in der Funktion genauso

unbekannt wie umgekehrt. Dadurch gibt es keine Konflikte mit verwendeten Namen. Außerdem wird eine weitere Forderung der Informatik erfüllt: Ein Block soll möglichst ein schwarzer Kasten (black box) sein, wobei oben Werte hineingetan werden und unten das Ergebnis herauskommt. Alles, was innerhalb dieser »black box« passiert, ist für das restliche Programm nicht vorhanden.

Bei der Textverarbeitung wurde Comal nicht an irgendeine Sprache angelehnt, sondern es wurde etwas eigenes entwickelt, was sich durch Klarheit und Geschwindigkeit besonders auszeichnet.

Zuerst einmal muß von vorneherein die maximale Länge jeder verwendeten String-Variablen angegeben werden. Dies geschieht mit dem Wort DIM, das aber ansonsten wie in Basic zum Anlegen von Arrays benutzt wird.

```
DIM name$ OF 20
DIM schüler$(0:50) OF 30
```

Dadurch wird der entsprechende Speicherplatz reserviert. Der Vorteil liegt auf der Hand: Es entstehen niemals »Müll-Strings«, und der Zugriff auf den Inhalt ist schnell, da sich die Lage besser errechnen läßt. Die Zeiten der gefürchteten Garbage Collection, dem zeitaufwendigen Ausräumen von String-Leichen, gehören der Vergangenheit an.

Zum anderen wird jeder String so behandelt, als wäre er das, was man in Pascal mit »Array Of Char« bezeichnet. Jedes einzelne Zeichen in einem String kann sofort adressiert und manipuliert werden.

```
DIM redaktion$ OF 40
redaktion$:="64er Redaktion"
PRINT redaktion$(6:8)
--->Red
PRINT redaktion$(:4)
--->64er
PRINT redaktion$(9:)
--->aktion
redaktion$(2):="5"
PRINT redaktion$
--->65er Redaktion
```

Dieses Konzept ist so flexibel, daß die Basic-Funktionen MID$, LEFT$ und RIGTH$ nicht mehr benötigt werden.

Natürlich hat diese Art von String-Verwaltung ihren Preis: Zum einen benötigt die Verwaltung dieser Strings zusätzlich zwei Byte, da neben der aktuellen Länge auch die dimensionierte Länge gespeichert wird. Zum anderen wird genau soviel Platz belegt, wie durch DIM reserviert wurde, auch wenn die Variablen leer sind. Aber es kann auch nicht mehr vorkommen, daß während eines Programmes die Meldung OUT OF MEMORY erscheint, nur weil die Strings zu sehr angewachsen sind.

Wie Sie gerade im letzten Beispiel gesehen haben, ist das Konzept der Teil-Strings sogar für Zuweisungen gültig:

```
INPUT "Nachname: ":eintrag$(16:30)
```

Die gesamte Eingabe wird in »eintrag$« ab der 16. Stelle abgelegt. Ist sie länger als die erlaubten 15 Zeichen (Stelle 16 bis 30), so wird der Rest einfach abgeschnitten. Ist sie kürzer, füllt Comal automatisch mit Leerzeichen auf. Überlegen Sie einmal die Einfachheit der Programmierung einer Dateiverwaltung.

Dynamische Fehlermeldungen

Ein wichtiger Punkt jeder Computersprache sind die verfügbaren Fehlermeldungen und der Umgang mit ihnen. Hier kann COMAL geradezu als Vorbild dienen.

Jede Zeile wird bei ihrer Eingabe zuerst einmal auf ihre syntaktische Richtigkeit geprüft. Eine fehlerhafte Zeile wird gar nicht erst angenommen. Spielen wir dies einmal durch:

```
10 FOR
```

Sofort nach dem <RETURN> meldet sich Comal mit »Schreibfehler«, und der Cursor steht direkt hinter dem FOR.

```
10 FOR i
```

Wieder wird der Schreibfehler gemeldet. Wir ergänzen:

```
10 FOR i=
```

Nun meckert Comal: »Ausdruck erwartet«.

```
10 FOR i=1
```

Auch hier gibt sich Comal nicht zufrieden. Dies setzt sich fort, bis die Zeile korrekt dasteht. Erst dann wird sie auch in den Speicher übernommen. Im Comal-Modul sind die Fehlermeldungen ausführlicher und klarer. So wird zum Beispiel direkt angegeben » := erwartet« oder »TO erwartet«.

Eine weitere Prüfung des Programmes findet statt, sobald Sie mit RUN das Programm starten. Hier werden die Strukturen getestet. Eine Prozedur kann nicht mit ENDFUNC abgeschlossen werden, ein ENDFOR ist bei REPEAT fehl am Platze, ein vergessenes ENDIF wird angemahnt und so fort. Dieser Check dauert auch bei langen Programmen weniger als eine Sekunde, wird also kaum bemerkt.

Die allerletzte Prüfung findet erst zur Laufzeit des Programmes statt. Hier werden solche Dinge wie undefinierte Texte oder arithmetischer Überlauf behandelt.

Wenn Sie mit dem MODUL-Comal arbeiten, so haben Sie zusätzlich noch komfortable Fehlerbehandlungsmöglichkeiten. Das nachfolgende Programmstück erlaubt es Ihnen, eine Zahl zwischen 1 und 50 einzugeben und dabei alle Fehlbedienungen auszuschließen:

```
LOOP
 TRAP
  INPUT AT 4,1,2:"Zahl (1..50) : ":zahl
  IF INT(zahl)<>zahl THEN REPORT
  IF (zahl<1) OR (zahl>50) THEN REPORT
  EXIT
 HANDLER
 ENDTRAP
ENDLOOP
```


Alles was in diesem Fall zwischen TRAP und ENDTRAP steht, ist in die Fehlerbehandlungsroutine eingebunden. Bei Auftreten eines Fehlers wird dem Benutzer der Fehler mit REPORT angezeigt. Zwischen HANDLER und ENDTRAP lassen sich noch weitere Anweisungen im Fall des Fehlerauftritts angeben. Auch eigene Fehlermeldungen lassen sich hier generieren, die dann wieder von übergeordneten behandelt werden können:

```
IF zahl>100 THEN REPORT 144, "Überlauf"
```

Wo erhält man Comal?

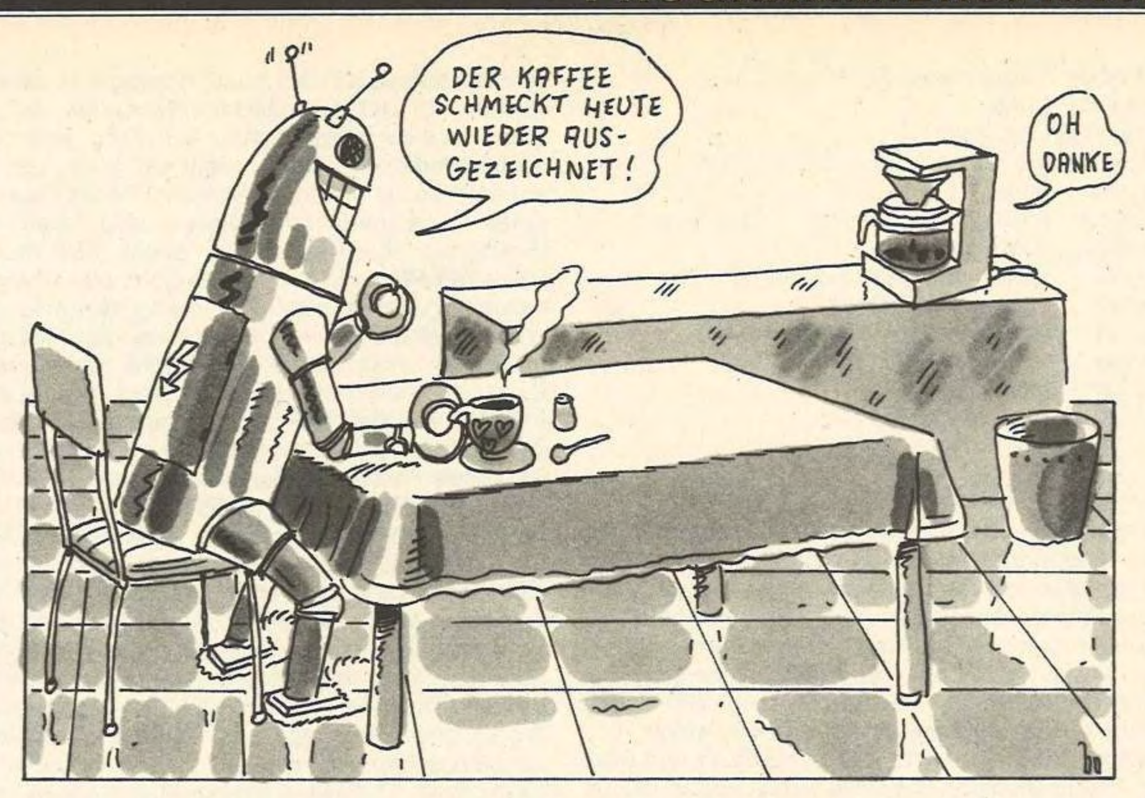
Comal gibt es für mehrere Rechner. Für den C64 sind momentan zwei Versionen verfügbar: Eine »freie« Version 0.14 und ein Modul, das Ihren Computer zugleich um 96 KByte erweitert. Die Bezugsquelle finden Sie am Ende des Artikels.

Im Gegensatz zu vielen anderen Sprachen wird nicht nur Comal gepflegt, sondern auch seine Verbreitung gefördert. Dies bedeutet, daß es viel freie Software gibt. Eine beidseitig bespielte Diskette für 15 Mark.

Die Comalgruppe Deutschland hat auch einen Telefon-Service eingerichtet. Hier werden alle Fragen zu Comal sofort geklärt.

Schließlich gehört auch noch eine Anwenderzeitschrift dazu, die in unregelmäßigen Abständen erscheint und voll ist mit Tips & Tricks, Neuigkeiten, Kursen etc. Der Anwender wird also nicht allein gelassen. (Siegfried Bauer/jk)

Bezugsquelle: Comal-Gruppe Deutschland, 2270 Utersum/Föhr, Tel. 04683-500, Mailbox 04683-554, Tel.-Service: 04627-543
Preis für das Comal-Modul: 209 Mark einschließlich deutschem Handbuch.
Diskette 0.14: 15 Mark. Handbuch: 6,50 Mark.



Prolog – die Sprache der künstlichen Intelligenz

Einen eigenen Bereich der Informatik bildet in der Zwischenzeit die KI-Forschung. Prolog ist die Sprache, die der KI-Forscher spricht, um seinem unwissenden Computer ein klein wenig Intelligenz beizubringen.

Um sich näher mit der Sprache Prolog zu beschäftigen, ist es ratsam, sich erst ein klein wenig Hintergrundwissen über die Entwicklung und Zielsetzung der künstlichen Intelligenz anzueignen. Womit wir schon beim größten Problem der ganzen KI-Forschung sind: Hintergrundwissen. Intelligenz setzt voraus, daß auf eine Situation an Hand von Erfahrungen und eben Wissen reagiert wird. Sie sind zum Beispiel ohne weiteres in der Lage, zwei Personen voneinander zu unterscheiden oder zu behaupten, daß diese Personen miteinander verwandt sind. Dazu benötigen Sie aber das Wissen, warum ein Verwandtschaftsverhältnis besteht. Dieses Wissen haben Sie sich irgendwann, beim ersten Kennenlernen, angeeignet. Genauso liegt das Problem beim Computer. Der Computer ist nicht in der Lage, logische Schlüsse zu ziehen, wenn nicht die benötigten Daten vorhanden sind. Der Mensch kann aufgrund der Tatsache, daß zwei Personen dieselben Eltern haben, die Schlußfolgerung ziehen, diese wären Geschwister. Der Mensch ist also aufgrund seines Wissens, manchmal auch als Lebenserfahrung bezeichnet, in der Lage, aus bestimmten Gegebenheiten eine passende und richtige Schlußfolgerung zu ziehen. Damit nun der Computer ebenfalls diesen Bildungsstand erreichen kann, muß ihm das Wissen, das Menschen über Jahre und Generationen gesammelt haben, erst mitgeteilt werden. Hierin liegt eines der Probleme der KI: die Speicherkapazität. Um alles Wissen der Menschen zu speichern, wären Unmengen an Speicherplatz vonnöten. Das zweite Problem ist die Geschwindigkeit eines KI-Programms. Ab einer gewissen Datenmenge tun sich herkömmliche Computer unheimlich schwer beim Durchsuchen dieser Datenbanken. Dazu werden in Japan die sogenannten »Computer der 5. Generation« entwickelt, die mit Parallelprozessoren arbeiten. Programmiert werden diese Super-Computer in Prolog, das sich die Japaner für ihre KI-Projekte entwickelt haben.

Regel- und Steuerwerk

Prolog kann nicht mit einer herkömmlichen Sprache verglichen werden, weder in der Arbeitsweise, noch im Sprachaufbau. Sicherlich kennen Sie aus Basic-Programmen die langen Ketten von nacheinander ablaufenden Befehlszeilen. Der Programmierer ist hier aufgefordert, die zu lösenden Probleme in die entsprechende Sprache umzusetzen. Er muß sich konkrete Algorithmen überlegen, mit denen das anstehende Problem gelöst werden kann. Nicht so in Prolog. Prolog verlangt zur Lösung einer Aufgabe nur die ausformulierte Problemstellung. Diese Problemstellung kann man sich als Kommentar vorstellen, wie Sie ihn sicherlich in Ihren Basic-Programmen verwenden. Diese Vorgehensweise wird auch als »deklarativ« bezeichnet. Im Gegensatz dazu nennt man den Aufbau herkömmlicher Programmiersprachen »prozedural«. Sie wissen also jetzt, daß Prolog eine kommentarorientierte Sprache ist. Doch wie kann man ein Programm aus ablauffähigen Kommentaren aufbauen? Sehen wir uns das Ganze näher an. Nehmen wir an, Sie wollten ein Programm, das dem Einsteiger sagen kann, ob beziehungsweise welche Computer für Prolog-Programmierung geeignet sind.

```
01)   ki_faehig (commodore64).
02)   ki_faehig (ibm).
03)   ki_faehig (ibm_comp).
04)   ki_faehig (vc20).
05)   ibm_comp  (ibm).
06)   ibm_comp  (commodorepc10).
07)   ibm_comp  (schneiderpc).
08)   program   (commodore64,prolog64).
09)   program   (ibm,turboprolog).
10)   program   (commodorepc10,turboprolog).
11)   program   (schneiderpc,turboprolog).
12)   ki_comp (X) if
13)         ki_faehig (X) and
14)         program (X,Y) or
15)         ibm_comp (X) and
16)         program (X,Y).
```

Dieses Programm sieht für den Prolog-Einsteiger völlig unleserlich aus. Man fragt sich, ob überhaupt ein Interpreter in der Lage ist, diese wirr erscheinende Zeichenfolge zu verstehen. Dem Prolog-Kenner präsentiert sich ein durchaus verständliches Programm mit klaren Regeln und Fakten. Damit wären wir auf einer weiteren Station unserer Reise durch Prolog angekommen, den Regeln und Fakten. Um diese Phänomene zu erklären, eignet sich das obige Programm bestens. In den ersten 11 Zeilen, die Nummern werden nur zur Dokumentation verwendet, erhält Prolog die für das Problem »Welcher Computer empfiehlt sich für die Programmierung in Prolog?« nötige Wissensbasis. Weder sind hier alle KI-fähigen Computer aufgezählt, noch die komplette Menge der Geräte, die auf dem Markt erhältlich sind. Dem Prolog-Interpreter wird mitgeteilt, daß der C 64, der IBM und alle IBM-kompatiblen KI-fähig sind. Danach erfolgt eine Aufzählung der IBM-kompatiblen Computer. Die Fakten mit dem Namen »program« geben nun noch die für die aufgeführten Computer erhältlichen Prolog-Produkte an. Damit haben wir Prolog alles Wissenswerte über Computer und zugehörige Prolog-Software mitgeteilt. Die Regel (Zeile 12-16) teilt dem Prolog-Interpreter die Bedingungen mit, wie er die Anwendereingabe mit der Wissensbasis, den Fakten, verknüpfen soll. Sehen wir doch eine mögliche Anwendereingabe an:

```
ki_comp (commodore64)
```

Die Variable »X« wird dabei fest an das Faktum »commodore64« gebunden. Das heißt, alle weiteren Operationen, die mit »X« durchgeführt werden, arbeiten von nun an mit »commodore64«. Ob der C 64 nun wirklich in der Lage ist, künstliche Intelligenz zu unterstützen, hängt davon ab, ob es entsprechende Software dazu gibt. Diese Fakten sind in den Programmzeilen zu finden, die mit »program« beginnen. Hier ist festgelegt, daß auf dem C 64 mit dem Programm »Prolog 64« KI-Programmierung möglich ist. Auf die obige Anfrage antwortet der Interpreter folglich mit »YES« oder »TRUE«. Interessant ist der Lösungsweg, den Prolog dabei geht.

Backtracking – Spaziergang durch das Wissen

Prolog ist eine sehr »ausdauernde« Programmiersprache. Stellen Sie sich vor, sie stehen in einem Irrgarten und wissen nicht mehr weiter. Was werden Sie tun? Nun, Sie gehen zur nächsten Wegkreuzung und wählen einen der möglichen Wege aus, in der Hoffnung, es sei der richtige. Landen Sie dann in einer Sackgasse, gehen Sie zurück zur letzten Kreuzung und versuchen einen anderen Weg. Das geht dann solange, bis Sie entweder den richtigen Weg gefunden haben oder es keine Möglichkeit mehr gibt, aus dem Irrgarten auszubrechen und Sie aufgeben. In diesem Stil arbeitet auch Prolog. Aus der Wissensbasis des obigen Programms ergeben sich im Zusammenhang mit der aufgestellten Regel mehrere Suchpfade für die Lösung. Prolog wird als erstes untersuchen, ob der eingegebene Computer »ki_faehig« ist. Dabei wird die Wissensbasis von oben nach unten durchsucht. Wird der Computer nicht gefunden, gibt Prolog logischerweise die Meldung »NO« aus. Findet Prolog den Computer im aufgeführten Wissen, wird noch die »AND«-Bedingung überprüft. Prolog sucht jetzt die »program«-Fakten ab. Wird dort der erfaßte Computer in Verbindung mit einem Programm gefunden, erfolgt die positive Meldung »YES«. Fällt die Abfrage negativ aus, gibt Prolog deswegen noch lange nicht auf. Vielmehr wird vom letzten Verknüpfungspunkt aus, der ein positives Ergebnis aufweisen konnte, der nächste mögliche Pfad durchgespielt. Von dort wird dann das nächste Faktum auf Richtigkeit getestet. Erst wenn alle »program«-Fakten mit negativem Ergebnis durchlaufen wurden, gibt Prolog auf und wendet sich dem nächsten Teil der Regel zu. Prolog versucht dort als erstes die IBM-Kompatibilität nachzuweisen und sucht dann bei positivem Ergebnis nach dem entspechenden Programm. Dieses Rückverfolgen und immer wieder neu von einem bestimmten, bereits vorher als richtig erkannten Punkt starten, nennt man »Backtracking«. Hierin liegt auch der große Unterschied zu konventionellen Datenbanken. Prolog ist im Gegensatz zu diesen Programmen in der Lage, einen falsch eingeschlagenen Weg zurückzugehen und den nächstmöglichen, falls vorhanden, auszutesten. Das Backtracking ist also eine der mächtigsten Eigenschaften von Prolog. Ohne das Backtracking wäre es auch nicht möglich, das zu programmieren, was man heute allgemein unter künstlicher Intelligenz versteht. Prolog wäre ohne diesen wichtigen Bestandteil wirklich nur eine Programmiersprache, um Datenbanken zu erstellen. Sie werden sich jetzt natürlich fragen, ob das schon alles ist. Natürlich kann Prolog noch einiges mehr.

Prolog und Listen

Aus Basic kennen wir dimensionierbare Felder, aus Pascal die »ARRAYS«. Prolog verwendet zur Darstellung von solchen Datenstrukturen Listen. In einer Liste können mehrere Angaben zu einem Prädikat gemacht werden. Nun, da der Begriff »Prädikat« endlich gefallen ist, wird er natürlich auch erklärt. Betrachten Sie doch noch einmal das obige Programm. Die Angaben über Computer und Programme stehen in Klammern. Davor steht der Name. Diese Kombination wird in Prolog als Prädikat bezeichnet. Eine Liste ist ebenfalls ein Prädikat, mit dem Unterschied, daß mehrere Begriffe eine Art Tabelle bilden. Sehen wir uns ein Beispiel für eine solche Liste an.

```
(Markt und Technik)
```

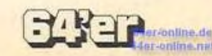
Diese Liste besteht aus drei Elementen. Prolog-Listen sind aber nicht auf drei Teile begrenzt, sondern können beliebig lang sein. Ein großer Vorteil gegenüber herkömmlichen Programmiersprachen. Es bleibt die Frage, wie diese Listen in Prolog behandelt werden. Am einfachsten sind da noch die Listen, die nur aus zwei Elementen bestehen. Diese werden über ein Listenmuster angesprochen.

```
(element1 element2)
```

In diesem Fall treten also keine Probleme auf. Schwieriger wird es, wenn die Liste eine beliebige Länge annimmt. Der Haken dabei ist, daß man oft selbst nicht weiß, wie lang die Liste nun wirklich ist. Prolog bietet auch dafür eine Lösung. Nehmen wir an, die Liste hätte, wie die bereits oben definierte, mehr als zwei Elemente. Um diese zu bearbeiten, verwendet man ein spezielles Listenmuster.

```
(element1  element2)
```

»element1« enthält jetzt den ersten Teil (erstes Element) und »element2« den Rest der Liste. Das Ganze sieht dann folgendermaßen aus:



```
element1 = Markt
element2 = und Technik
```

Sie wissen jetzt, daß Listen eine beliebige Anzahl an Elementen haben können und daß sie mittels Listenmustern bearbeitet werden. Listen können aber auch beliebig viele Sublisten enthalten. Sehen wir uns eine solche »verschachtelte« Liste an:

```
(((Markt und Technik)
(Buchverlag Softwareverlag)
(Homecomputer Personalcomputer)))
```

Daraus kann jetzt ein dreielementiges Listenmuster gebildet werden, um die Liste anzusprechen.

```
(Verlag Branche Zielgruppe)
```

Als Ergebnis der daraus folgenden Verknüpfung erhalten wir ein eindeutiges Ergebnis:

```
Verlag     = Markt und Technik
Branche    = Buchverlag Softwareverlag
Zielgruppe = Homecomputer Personalcomputer
```

Sie sehen, die Listen in Prolog sind durch die verschiedenen Listenmuster, mit denen sie angesprochen werden können, sehr flexibel und können nicht nur als Aneinanderreihung von Daten Verwendung finden, sondern durch die beliebige Unterteilung mit Sublisten auch zur Strukturierung von Daten eingesetzt werden.

Arithmetik und Rekursion

Bis jetzt war immer von Datenmanipulation die Rede. Es wurde aufgezeigt, wie flexibel Prolog bei der Handhabung von großen Datenmengen ist. Man kann aber mit Prolog auch rechnen. Allerdings funktioniert das ganz anders, als Sie das vielleicht von herkömmlichen Programmiersprachen gewohnt sind. Für die Arithmetik gibt es in den meisten Prolog-Versionen ein eingebautes Prädikat: »SUM«. Wie gesagt, die anschließend geschilderte Methode ist sehr ungewohnt, aber auch ebenso praktisch. Betrachten wir doch einmal eine Möglichkeit der Addition:

```
SUM (A,B,C)
```

Diese Aussage ist solange richtig, wie die Summe aus A und B gleich C ist. Wenn Sie diesen Ausdruck in Ihr Programm einbauen, können Sie ihn folgendermaßen aufrufen.

```
SUM(7,8,15)
```

Da sieben plus acht fünfzehn ergibt, antwortet Prolog prompt mit »YES«. Sollten Sie Zahlen eingeben, die nicht mit dem Ergebnis (»C«) übereinstimmen, erhalten Sie natürlich ein »NO« als Antwort. Mit diesem Prädikat kann jetzt auch subtrahiert werden. Sie wollen beispielsweise die Differenz aus »18 – 10« feststellen. Die Subtraktion kann ohne weiteres auf die Addition zurückgeführt werden. Das bedeutet, daß 18 die Summe aus 10 und einer Unbekannten ist. Darüber brauchen wir uns keine Gedanken zu machen, diese Arbeit übernimmt Prolog.

```
SUM (A,10,18)
```

Prolog antwortet jetzt, wer hätte es anders erwartet, mit acht. Vielleicht haben Sie jetzt schon erkannt, daß es eine Möglichkeit geben müßte, unendlich viele Ergebnisse anzufordern.

```
SUM (A,B,C)
```

Das ist nicht eine Wiederholung der obigen Definition. Prolog sollte jetzt einfach alle möglichen Kombinationen ausgeben, die ein Ergebnis aufweisen. So fatal das klingen mag, es würden hier logischerweise alle Additionen durchgespielt, die überhaupt möglich sind. Hier stößt man schnell an die Grenzen von Prolog und des verwendeten Computers. Laut theoretischem Sprachstandard ist diese Kombination ohne weiteres möglich. Doch dazu müßte die Prozessorlogik ebenso deklaritiv aufgebaut sein wie Prolog. Die verwendeten Prozessoren in Heim- und Personal Computern arbeiten heute aber mit prozeduraler Prozessorlogik. Vielleicht bringen es die Japaner mit ihrem ehrgeizigen Projekt, den Computer der 5. Generation zu bauen, einmal fertig, solch komplexe Anfragen an ein Prolog-Programm beantworten zu können.

Kommen wir aber jetzt noch einmal auf die bereits behandelten Listen zu sprechen. Bisher hatten Sie nur Zugriff auf das erste Element, während der Rest komplett als zweites Element verarbeitet wird. Listen können in Prolog rekursiv angewendet werden. Dabei wird die komplette Liste einfach umgedreht. Das letzte Element ersetzt das erste und das erste das letzte. In der Theorie klingt das wieder unheimlich schwierig. Prolog stellt für Listenmanipulation wieder zwei Standardprädikate zur Verfügung: REVERSE und APPEND, wobei APPEND nichts mit Anhängen zu tun hat.

```
reverse ((Element1 Element2),
        rekursive_Liste)
```

Die bearbeitete Liste wird wieder in das erste Element und den Rest aufgeteilt. Danach dreht Prolog die gesamte Liste um (Rekursion). »Element2« kann dann ebenfalls wieder umgekehrt werden und so weiter.

Ein etwas anderes Befehlsformat weist APPEND auf. Dieses vordefinierte Prädikat findet beim Listenvergleich Verwendung. Zwei Listen werden zusammengebunden und das Ergebnis mit einer dritten verglichen.

```
append((64er),(Magazin),(64erMagazin))
```

Diese Eingabe wird Prolog mit Sicherheit durch ein »YES« bestätigen. Sie sehen, daß Prolog von vorneherein einige nützliche Prädikate eingebaut hat, die für ein sinnvolles Programmieren unerläßlich sind.

Daß Prolog nicht gerade eine Sprache zum Erstellen von herkömmlichen Datenbanken ist, haben Sie sicherlich schon festgestellt. Auch für komfortable Dialogprogramme nimmt man doch besser die bekannten und leistungsfähigen Sprachen, wie etwa Pascal, her. Wem sollte also dieses futuristisch anmutende Prolog nützen? Nun, da gibt es natürlich einmal die KI-Forscher, die versuchen, den Computern einen Hauch von menschlicher Vernunft beizubringen. Da ist auf der anderen Seite das militärische Interesse an künstlicher Intelligenz und da gibt es die Industrie, die alles immer noch besser und noch schneller machen will.

Prolog-Anwendungen

Die Industrie kann sich dazu heute verschiedener Prozeßsteuerungen bedienen, die bereits in Prolog geschrieben sind. Diese Programme sind in der Lage, Veränderungen an Werkstücken selbständig zu erkennen und eine entsprechende Reaktion zu zeigen. So gibt es beispielsweise Drehmaschinen, die, je nach Werkstück und Beschaffenheit, das benötigte Werkzeug auf eigene Veranlassung einlegen und damit arbeiten.

Ein weiteres Beispiel für künstliche Intelligenz stammt aus den USA. Wenn das System auch nicht in Prolog programmiert wurde, sondern in der von den Amerikanern verwendeten KI-Sprache LISP, demonstriert es doch deutlich die momentanen Fähigkeiten der KI. Dieses Programm wurde für das Analysieren von Bodenproben eingesetzt und sollte aus den Angaben, die über die Probe gemacht wurden, erkennen können, ob sich in der Gegend, aus der die Probe stammte, Mineralstoffvorkommen befinden. Nach Aussagen der Amerikaner wurde mit Hilfe dieses KI-Systems ein großes Vorkommen entdeckt und zwar ohne die Probe monatelang zu untersuchen.

In diesem Fall spricht man von Expertensystemen, die in der Wirtschaft seit längerer Zeit professionell eingesetzt werden. Diese Systeme verfügen über eine große Wissensbasis zu einem bestimmten Fachgebiet, das jederzeit abrufbar und

in der Lage ist, Fragen zu diesem Fachgebiet zu beantworten. Der größte Suppenhersteller in den USA setzt ein solches Expertensystem in Verbindung mit seinem Maschinenpark ein. Der »Experte« Computer gibt dem Techniker bei Produktionsstörungen konkrete Hinweise auf die Fehlerquelle und bietet gleichzeitig eine oder mehrere Lösungen an.

Natürlich muß dieses Wissen dem Computer erst zur Verfügung gestellt werden. Dazu bildete sich in letzter Zeit das Berufsbild des »Knowledge Engineer« (Wissens-Ingenieur). Er hat die Aufgabe, sich das Wissen über ein Fachgebiet zu verschaffen und die Umsetzung dieses Wissens auf den Computer vorzubereiten. Das Wissen erhält er dabei von einem erfahrenen Fachmann.

Die meisten auf dem Markt erhältlichen Expertensysteme sind heute nicht nur viel zu teuer, sondern auch nur für Großrechner erhältlich. Außerdem beschränken Sie sich größtenteils auf bestimmte Fachgebiete. Die Bestrebungen gehen in Richtung flexibles Expertensystem. Diese Systeme bieten dem Anwender ein Grundprogramm, in das er sein spezielles oder globales Wissen eingeben kann. Dieses steht konstant zur Verfügung. Denkbar sind (theoretisch) Datenbanken, die das gesamte Wissen der Menschheit gespeichert haben. Wenn Sie als Privatmann dann von zu Hause auf diese Daten zugreifen können, dürfte nicht nur ein Traum mancher KI-Forscher erfüllt sein. Dann wird sich auch die totale Informationsgesellschaft gebildet haben.

Damit sind die Anwendungsbereiche der KI allerdings noch nicht erschöpft. Ein weiteres KI-Problem ist die Bilderkennung. Wenn der Mensch einen Vogel sieht, weiß er eben, daß es sich um einen Vogel handelt. Zwar ist das Sehen für Computer durch die Digitalisiertechnik heute kein Problem mehr, die Schwierigkeit besteht in der Herstellung des Bezugs zum Bild. Der Computer kann nicht ohne weiteres feststellen, daß es sich um eine bestimmte Sache handelt, wenn nicht als Gegenstück die Sache zum Vergleich im Speicher bereitsteht. Da jedoch bewegliche Körper verschiedene Zustände eingehen können, ist die Schwierigkeit dieser Aufgabe für den Computer leicht abzuschätzen.

Ein letztes Arbeitsgebiet der KI-Forscher ist die Sprache. So arbeiten die Japaner und Amerikaner darauf hin, dem Computer Ohren zu verleihen. Dann wäre es tatsächlich möglich, sich mit dem Computer zu unterhalten. Diese Errungenschaft liegt zwar noch in weiter Ferne, doch ist es bereits möglich, mit einem Prolog-Programm in geschriebener Sprache zu kommunizieren. Allerdings sollten sich nur Profis an ein solches Projekt heranwagen.

Was bringt die Zukunft?

Die KI-Forschung steht heute an der Schwelle zur »mechanischen Intelligenz«. Computer sollen für den Menschen das Denken übernehmen. Das ist zwar noch Zukunftsmusik, soll aber möglichst schnell realisiert werden. Es ist die Aufgabe des Menschen, die Entwicklung stets unter Kontrolle zu halten. Es liegt in der Hand der Bewohner dieses Planeten, die Errungenschaften der KI für die Verbesserung der vorherrschenden Lebensbedingungen einzusetzen. Auf keinen Fall darf die Maschine den Menschen ersetzen oder zu einer geistigen Verarmung führen, was bei einem »denkenden« Computer nicht mehr auszuschließen ist. Wer sich näher mit KI befaßt, fühlt sich manchmal erschreckt an den gläsernen Menschen oder den »Großen Bruder« erinnert. Trotz der Faszination, die KI auf den Computer-Freak ausübt, muß auch auf die Gefahren dieser Wissenschaft hingewiesen werden. Wie gesagt, der Mensch hat es in der Hand. (rf)

C – Die Sprache des Systemprogrammierers

Für Programmierer, die sich nicht mit Maschinensprache abmühen wollen, aber trotzdem gerne auf Systemebene zugreifen würden, ist C genau das richtige. Strukturiert und maschinennah präsentiert sich diese Sprache dem Anwender.

Bisher konnte sich der begeisterte Computer-Freak noch mit seinen Basic- und Assembler-Kenntnissen durchschlagen. Bis vor einiger Zeit die Sprache C auf dem Software-Markt auftauchte. Von C hörte man Dinge, die das Herz jedes Programmierers höher schlagen lassen. Da ist zum einen die strukturierte Programmierung, zum anderen die Möglichkeit, maschinennah zu programmieren. Doch was verbirgt sich hinter C? Wo liegen die Vorteile gegenüber Basic, Pascal und anderen geläufigen Sprachen, von KI-Exoten und Spezialentwicklungen abgesehen?

C entstand während der Versuche der amerikanischen Bell-Laboratories, das Betriebssystem Unix zu entwickeln und auf verschiedene Computer umzusetzen. UNIX ist ein extrem leistungsfähiges Mehrbenutzer-(Multiuser) und Multitasking-Betriebssystem, das bis dahin fast ausschließlich in Assembler geschrieben war und dementsprechend wenig Verbreitung fand, denn es mußte für jeden Prozessor neu entwickelt werden. Diese Probleme plagten den Programmierer Dennis M. Ritchie von den Bell-Laboratories und er schuf – C. Es sollte eine übertragbare, schnelle, also maschinennahe und standardisierte Hochsprache werden, mit allen Strukturbefehlen, wie man sie eigentlich nur von unstandardisierten, häufig recht langsamen Systemen her kannte. Als erstes wurde natürlich das Betriebssystem UNIX in C neu geschrieben. Seither ist Unix auf vielen Computern verfügbar.

Der Sprachschatz von C ist standardisiert. Es entfallen die aus Basic bekannten Schwierigkeiten mit vielen Dialekten, die nichts mehr gemein haben. So können C-Programme auf kleinen, billigen Terminals entwickelt und in nahezu unveränderter Form auf den teuren Supercomputern eingesetzt werden.

C-Programme zeichnen sich auch durch eine hohe Geschwindigkeit aus, vor allem, wenn von der Möglichkeit der maschinennahen Programmierung Gebrauch gemacht wird. So können beispielsweise mit einem einzigen Befehl Bits rotiert werden, ebenso leicht können einzelne Bit eines Byte direkt angesprochen werden. Außerdem ist es möglich, häufig benutzte Werte in Prozessorregistern abzulegen (zum

Beispiel Schleifenindizes), was den Vorteil bringt, daß der Prozessor direkt, ohne Umwege über Adreßschiebereien, auf die Daten zugreifen kann. Diese Vorteile dürften sicher auch so manchen Assembler-Freak dazu veranlassen, sich mit C zu beschäftigen.

Der einzige Standard

Nun zu einer Eigenheit von C, die den Basic-Benutzer anfangs zumindest entsetzen wird: die Programmeingabe. In Basic sieht es (bei einfachen Versionen) noch so aus:

```
Zeilennummer...Befehl1:Befehl2:...
```

Außerdem hat man den Direktmodus, Basic ist eine Interpretersprache, über den vieles vor dem Einbau ins Programm getestet werden kann. Fehlermeldungen werden – während des Programmlaufs – oft in rauhen Mengen auf uns losgelassen.

C dagegen ist eine Compiler-Sprache. Bei nahezu allen C-Systemen wird das Programm über einen speziellen Editor eingegeben, der oft viele Funktionen einer professionellen Textverarbeitung besitzt. Dabei wird das Programm an einem Stück eingegeben, beziehungsweise einzelne Module werden aneinandergehängt. Erhält der Programmtext das Prädikat »wertvoll«, kann man mit Schritt zwei beginnen: dem Compilieren. Hier wird das erstellte Programm in Maschinensprache übersetzt. Abschließend wird noch ein Link-Vorgang durchgeführt. Dabei wird um das eigentliche Programm noch ein Paket aller benutzten System- und sonstigen Unterroutinen gebastelt, so daß das Programm auch ohne die C-Umgebung lauffähig ist. Bei manchen C-Compilern wird beim Compilieren Assembler-Code erzeugt, das heißt, das Programm muß vor dem Linken noch zusätzlich assembliert werden. Bei kurzen Programmen gestaltet sich dieser »Edit-Compile-Link«-Zyklus noch erträglich. Doch bei größeren Gebilden sind Compilier- und Linkzeiten von mehreren Minuten keine Seltenheit, sondern die Regel. Befinden sich dann noch Fehler im Programm, muß der ganze Vorgang neu durchlaufen werden. Deshalb sollte der Programmierer darauf achten, daß der Quelltext gut vorbereitet wurde. Ausführliche Vorarbeiten sind vor der eigentlichen Programmierung notwendig.

Edit-Compile-Link-Run

Die Ablaufgeschwindigkeit der Programme wiegt diesen Nachteil wieder auf. C unterscheidet sich auch noch in anderen Punkten von herkömmlichen Programmiersprachen. So hat man beispielsweise nicht alle Befehle im eigentlichen Sprachkern (dieser umfaßt lediglich 28 Befehle (Tabelle 1), wie beispielsweise die Anweisungen für Schleifenkonstrukte und Variablendefinitionen. Alles andere, wie etwa Befehle zur Bildschirmausgabe, wird in Form von Systembibliotheken auf Diskette mitgeliefert. Das Konzept liegt auf der Hand: Kompatibilität. Denn alles, was auf die Maschine direkt zugreift, wird ausgelagert und nur bei Gebrauch in das Programm eingebunden. Jeder Computer hat unterschiedliche Betriebssystemroutinen, für die dann die Systembibliotheken angepaßt werden müssen.

Doch nun zu den weiter oben bereits angesprochenen Variablendeklarationen. In Basic kennen wir (in dieser expliziten Form) solche Vereinbarungen eigentlich nicht. Dort kann einfach mitten im Programm eine Variable verwendet werden, ohne daß diese vorher irgendwie gesondert definiert worden wäre. C jedoch verlangt, ähnlich wie Pascal, die Deklaration jeder Variablen vor ihrer Benutzung. Eine solche Definition sieht dann beispielsweise folgendermaßen aus:

```
int a
int a,b
```

Die 28 C-Schlüsselwörter				
auto	break	case	char	continue
default	do	double	else	entry
extern	float	for	got	if
int	long	register	return	short
sizeof	static	struct	switch	typedef
union	unsigned	while		

Tabelle 1. Die Standard-Schlüsselworte von C

Im ersten Beispiel wird eine Integer (Ganzzahlvariable) mit dem Namen »a« vereinbart. Ebenso im zweiten Beispiel, nur zusätzlich noch eine Variable »b«. Integer werden oft in zwei Byte gespeichert und meist vorzeichenbehaftet verwendet. Integer, wie sie oben definiert wurden, erstrecken sich »nur« über den Bereich zwischen -32868 und 32867. »Long integers« werden genauso definiert wie die gewöhnlichen, nur mit vorangestelltem »long«.

```
long int c,f,g
```

Diese Zahlen (sie belegen doppelt soviele Bytes als die normalen Ganzzahlvariablen) haben eine »Bandbreite« von -2147483648 bis 2147483647.

Es können auch »short integers« (sie belegen 1 Byte), vorzeichenlose Zahlen (»unsigned«), Zeichenvariable (»char«) und Fließkommazahlen (»float«) vereinbart werden.

Dem Basic-Programmierer mag es unverständlich und umständlich erscheinen, jede Varible vor der Verwendung definieren zu müssen. Doch hinter diesem scheinbaren Nachteil verbirgt sich eine große Hilfe. Jeder, der umfangreichere Basic-Programme schreibt, wird zugeben, daß er es mit der Eindeutigkeit von Variablen nicht so genau nimmt. Hier ist »E« mal Schleifenindex, mal steht es als Variable zur Berechnung der Einwohner in Hinterdupfing. Und am Ende hat »E« dann einen Wert, den es gerade nicht hätte haben dürfen. Bei C weiß man eben immer, welche Variable wofür verwendet wird. Außerdem lassen sich lokale Variable definieren, also solche, die nur für ein spezielles Unterprogramm verwendet werden und im Hauptprogramm keine Verwendung finden können.

C ist eine strukturierte Hochsprache. Der Programmierer hat also alle Möglichkeiten zur strukturierten und modularen Programmierung. Es gibt die IF..ELSE-Struktur, FOR-, WHILE-, DO..WHILE- sowie SWITCH..CASE-Konstruktionen. Aus dieser Auswahl sehen wir uns die IF..ELSE-Anweisung näher an. In C lautet das allgemeine Format:

```
if (logischer Ausdruck) Befehl 1
                        else Befehl 2
```

Wo bleibt das von fast allen anderen Programmiersprachen her bekannte »then«? Wir brauchen es ganz einfach nicht. »Befehl 1« wird ausgeführt, wenn der logische Ausdruck wahr ist, ansonsten tritt »Befehl 2« in Aktion. Hierbei zeigt sich ein weiterer Vorteil von C-Programmen. Befehle tragen keinen unnötigen Ballast mit sich herum, alles kann kurz und einfach umschrieben werden. Doch zurück zu IF..ELSE. Ein kleines Beispiel könnte so aussehen:

```
if (a<b) c=a
         else c=b
```

Wenn also »a« kleiner ist als »b«, dann wird »c« mit »a« gleichgesetzt, sonst wird der Befehl nach ELSE ausgeführt (c=b). So einfach kann das sein.

In C braucht man kein umständliches »x=x+1«, um die Variable x zu inkrementieren (um 1 zu erhöhen), dafür hat man spezielle Befehle. So hat »++x« genau dieselbe Wirkung wie die oben angeführte Variablendreherei. Ähnlich funktioniert auch das Dekrementieren mit »--x«.

Doch Vorsicht ist gerade bei dieser Kürze der Anweisungen geboten! Schnell schreibt man »x++«, was auch einem gültigen Befehl entspricht, aber eine andere Funktion hat als



Grund genug fürs neue 64'er

In der Januar-Ausgabe stellen wir vor,

Monitore:

- Große Marktübersicht
- Professionelles Testbild für Monitore zum Abtippen
- ... und wichtige Informationen zum Monitor- bzw. Fernseherkauf.

Spielefans aufgepaßt!

- Zahlreiche Kaufhilfen für Computerspiele.
- Die besten Spiele des Jahres werden gesucht. 99 tolle Spiele können Sie dabei gewinnen.
- Zwei Familienspiele im Test.

Umfangreicher Einsteigerteil:

Viele Hinweise und Tips zur Schleifenprogrammierung

... außerdem lesen Sie:

■ Grafik-Ram für den MPS802: das neue Betriebssystem ermöglicht u.a. Grafik mit 640 Punkten pro Zeile ■ Das Jahresinhaltsverzeichnis: Damit Sie Artikel, Listings und Testberichte besonders schnell finden können ■ Das Listing des Monats: Leistungsstarkes Dame-Spiel zum Abtippen ■ Bedienung im Fachhandel: Unsere Tester berichten ihre Erfahrungen im Computerfachhandel, mit Fernseh- und Radiogeschäften- bzw. Abteilungen ■ Die Drucker Seikosha SL-80AI und Brother HR-10C überzeugen durch hohe Druckqualität und niedrigen Preis.

Falls Sie »64'er« noch nicht regelmäßig beziehen, sichern Sie sich jetzt Ihr persönliches Abonnement und nutzen die damit verbundenen Vorteile: ■ Sie beziehen »64'er« ohne Mehrkosten bequem per Post frei Haus ■ Sie haben Ihr »64'er« bereits bei sich zu Hause — noch bevor Sie es bei Ihrem Zeitschriftenhändler kaufen können. ■ Sie sind sicher, keine Ausgabe zu versäumen.

Sie erhalten — wenn Sie zur Anforderung den nebenstehenden Gutschein verwenden — auf alle Fälle die neueste Ausgabe als Probeheft unverbindlich und kostenlos.

Gutschein

FÜR EIN KOSTENLOSES PROBEEXEMPLAR DES 64'er-MAGAZINS

JA, ich möchte »64'er«, das Magazin für Computerfans, kennenlernen.
Senden Sie mir bitte die aktuellste Ausgabe kostenlos als Probeexemplar. Wenn mir »64'er« gefällt und ich es regelmäßig weiterbeziehen möchte, brauche ich nichts zu tun: Ich erhalte »64'er« dann regelmäßig frei Haus per Post und bezahle pro Jahr nur DM 78,— (Ausland auf Anfrage).

Vorname, Name

Straße | PLZ, Ort

Datum | 1. Unterschrift

Mir ist bekannt, daß ich diese Bestellung innerhalb von 8 Tagen bei der BesteIladresse widerrufen kann und bestätige dies durch meine zweite Unterschrift. Zur Wahrung der Frist genügt die rechtzeitige Absendung des Widerrufs.

Datum | 2. Unterschrift

Gutschein ausfüllen, ausschneiden, in ein Kuvert stecken und absenden an:
Markt&Technik Verlag Aktiengesellschaft, Vertrieb, Postfach 1304, 8013 Haar

64S1286

»++x«. Auch sonst macht der Compiler bis auf Syntaxprüfungen keine weiteren Fehlerabfragen; und zwar nicht, um dem Programmierer die Fehlersuche in seinen Produkten zu erschweren, sondern um ihm bei der Art der Anweisungen freie Hand zu lassen. Es gibt sogar Situationen, in denen der Anfänger keinen Zusammenhang in den Anweisungen eines Programmes feststellen kann. Für den Profi jedoch mag gerade dies das »non-plus-ultra« darstellen.

Ein weiteres, fast lebenswichtiges Werkzeug für den Programmierer sind Pointer (=Zeiger). Es soll auf etwas gezeigt werden. Im speziellen C-Fall gibt es beispielsweise Variablenpointer, über die man nicht den Inhalt der Variablen erhält, sondern direkt deren Adresse im Speicher. Was dadurch in Verbindung mit Bit-Manipulationen aus dem Computer herausgeholt werden kann, läßt sich erahnen.

Was bietet C aber nun dem Maschinen-Freak? Die eben angesprochenen Pointer und auch die wichtigsten Bit-Befehle, die schließlich die maschinennahe Programmierung erst ermöglichen. So gibt es Befehle zum Shiften (Verschieben) der Bits in einem Byte natürlich in beide Richtungen, sowie und-/oder-/exklusiv-oder-Verknüpfungen und Einerkomplementbildung. In einem Beispiel sieht das folgendermaßen aus:

```
*b<<
```

wobei in »b« eine Adresse für einen Speicherplatz steht, mit »*« der Zugriff auf diesen »freigegeben« und schließlich die Bits einmal nach links geshiftet werden (was einer Multiplikation mit zwei entspricht). Zeigen die spitzen Klammern in die andere Richtung, so werden die Bits einmal nach rechts geshiftet, was einer Division durch zwei entspricht. Der Operator für bitweises »und« heißt <&>, für »exklusiv oder« <^>, für »oder« <I>; ein <~> bildet das Einerkomplement einer Zahl, was einem Umkehren aller Bits entspricht. Soviel zu den Elementen, die wohl hauptsächlich den Assembler-Programmierer interessieren werden.

Inzwischen dürfte es klar herausgekommen sein: C ist unglaublich flexibel und leistungsstark. Wesentlich leistungsfähiger sogar als jede andere geläufige, höhere Programmiersprache. Um auch einen C-Standard-Befehl zu erklären, sehen wir uns die PRINTF-Anweisung näher an. An dieser Anweisung läßt sich auch gut das durchdachte Konzept der Bibliotheksroutinen sowie die Flexibilität der Sprache demonstrieren.

Ein letzter Vergleich mit Basic sei gestattet:

```
PRINT "1234ABC"
```

gibt die Zeichen zwischen den Anführungszeichen auf dem Bildschirm aus. Formatierte Zahlenausgabe ist im Basic V2.0 des C 64 nicht über einen PRINT USING-Befehl, wie ihn viele andere Basic-Dialekte kennen, möglich. Man muß eigene Ausgaberoutinen in Assembler schreiben, wobei oft mit viel Aufwand recht gute Ergebnisse zu erzielen sind. Doch sind diese Routinen dann auch flexibel? Was macht der Programmierer, der in Assembler noch nicht so bewandert ist?

C bietet genügend Lösungen für diese Probleme. Das Zauberwort heißt PRINTF und ist eine Abkürzung für »print formattet«, was soviel bedeutet wie formatierte Zeichenausgabe. PRINTF ist nicht im Sprachkern enthalten, sondern in einer der Standard-Bibliotheken untergebracht. Gerade deshalb konnte dieser Befehl so vielseitig gestaltet werden.

Nach PRINTF folgt von Klammern eingeschlossen eine Umwandlungsvorschrift, die auch als Kontrollstring bezeichnet wird. Danach folgt, vom Rest durch Komma getrennt, die auszugebende Zahl. Anhand eines Beispiels läßt sich das veranschaulichen.

```
printf ('%5d',z)
```

Der Kontrollstring beinhaltet »%«,»5« und »d«. Das »%«-Zeichen kennzeichnet den Kontrollstring, also fängt jeder mit diesem an. Die »5« veranlaßt den Computer zur Reservierung eines (5 Zeichen langen) Leerzeichenfeldes. Das »d« schließlich gibt an, wie die Zahl umgewandelt werden soll, bevor sie rechtsbündig in das Leerzeichenfeld geschrieben wird. Reicht die angegebene Anzahl der Leerzeichen nicht aus, werden automatisch weitere angehängt. Die Zahl, welche die Anzahl der reservierten Leerzeichen festlegt (auch Feldbreite genannt), kann auch entfallen; in diesem Fall wird automatisch die richtige Feldbreite errechnet und verwendet. Außer »d« (für dezimale Integerzahl) stehen noch viele weitere Umwandlungsanweisungen zur Verfügung (Tabelle 2). In Listing 1 sehen Sie noch ein kurzes C-Programm, das einige der hier angesprochenen Befehle verwendet. Dabei gibt es noch eines anzumerken: Das Hauptprogramm beginnt immer mit »main ()«. Das eigentliche Programm wird von geschweiften Klammern umrahmt.

```
/* Programm zur Berechnung der Quadrate der Zahlen von 1 bis 100*/

main()
{
     int z,q;

     printf("Dieses Programm berechnet das Quadrat der Zahlen 1-10");
     for (z = 1; z <= 10; ++i)   /* Schleife von 1 bis 10 */
     {
         q = z * z;              /* Quadrieren */
         printf ("%d",z,q);      /* Formatierte Ausgabe */
     }
} /* Ende Programm */
```

Listing 1. Beispielprogramm in C

Formatanweisungen für den PRINT-Befehl	
ld	lange Integer
lo	lange Oktalzahl
lx	lange Hexadezimalzahl
b	ganzzahlige Binärzahl
c	einzelnes Zeichen
d	ganzzahlige Dezimalzahl
e	Fließkommazahl mit Exponent
f	Fließkommazahl in voller Länge ohne Exponent
g	verwendet entweder e oder f, je nach Speicherbedarf
h	kurze Integer
o	ganzzahlige Oktalzahl
s	Zeichenkette
u	vorzeichenlose Integer
x	ganzzahlige Hexadezimalzahl

Tabelle 2. Formatanweisungen

Doch wofür kann man diese offensichtlich einmalige Programmiersprache überhaupt einsetzen? Beispiele lassen sich genügend anführen. Durch geschickte Programmierung lassen sich, beispielsweise bei Grafikprogrammen, Geschwindigkeiten erzielen, die nahe an die reiner Assembler-Programme heranreichen, jedoch durch wesentlich übersichtlicher strukturierte und vor allem portable Programmierung erstellt wurden. Außerdem ist ein Paradebeispiel der Amiga. Das Betriebssystem dieses Computers ist in C geschrieben.

Auch viele Universitäten kommen von den althergebrachten Sprachen (Cobol, Fortran) langsam ab und wenden sich immer mehr C zu.

Lohnt es sich also, für den Heimcomputer einen C-Compiler zu beschaffen? Wer nicht erwartet, daß diese Sprachumsetzungen all das bieten, was für PCs und Minicomputer verfügbar ist und trotzdem die Programmierung unter C lernen möchte, ist sicherlich gut beraten, sich einen Compiler zu leisten. C hat Zukunft eingebaut. (Ingolf Krüger/rf)

Freesoft-Forth: Die starke Alternative

Software zum Nulltarif ist der Traum eines jeden Computerbesitzers. Jetzt können Sie sich die leistungsfähige Programmiersprache Forth auf diesem Weg für den C64 erschließen.

Die Programmiersprache Forth ist jetzt in der Version Ultraforth 83 für den C64 erhältlich. Der Clou daran ist nun jedoch, daß die Forth-Systemsoftware als Freesoft veröffentlicht wurde. Das heißt, man kann in den Besitz dieser Programmiersprache kommen ohne viel Geld auszugeben. Eine kleine Begriffserläuterung wäre im Zusammenhang mit Freesoft allerdings angebracht. Public Domain (PD) bedeutet frei übersetzt »für die Öffentlichkeit«. Auf den Computer angewandt, heißt PD nichts anderes, als daß es Programmautoren gibt, die, wenn sie ein Programm entwickeln, ihr Wissen und Können fast kostenlos zur Verfügung stellen. PD-Programme können zum Selbstkostenpreis bezogen werden, sind frei kopierbar und dürfen an andere Computerbenutzer weitergegeben werden. Freesoft und Shareware sind Ableger des PD-Bereichs, mit denen der Anwender genauso verfahren kann, jedoch meist im Programmvorspann gebeten wird, bei intensiver Nutzung des Programms einen geringen Geldbetrag an den Autor zu überweisen. Oft erhält man dann eine neuere Programmversion und/oder eine ausführliche Programmdokumentation. Angeboten werden auf diesem Weg hauptsächlich Utilities, also kurze Hilfsprogramme, Spiele sowie seit einiger Zeit auch Programmiersprachen. Und eine dieser Programmiersprachen, genauer Ultraforth 83, das uns in der Version 3.5 zum Test vorlag, wurde von Mitgliedern der Forth-Gesellschaft e.V. für den C64 entwickelt. Das Forth-System ist auf zwei beidseitig beschriebenen Disketten erhältlich, auf denen sich Forth, Erklärungstexte, Hilfsprogramme, Quelltexte, ein Assembler und Grafikroutinen befinden. Weiterhin lag uns auch noch ein Handbuch als Ringordner vor, das nach Aussagen der Autoren ständig erweitert wird und später sogar über den Buchhandel erhältlich sein soll. Ultraforth 83 ist eine wirklich fantastische Umsetzung der Sprache auf dem C64, die nahezu keine Wünsche offen läßt. Auch für andere Computer sind bereits Umsetzungen geplant bzw. erhältlich.

Multitasking integriert

Geladen wird Forth von der »Systemdiskette« auf der auch eine Demonstration sowie verschiedene Hilfsbildschirme (Screens) enthalten sind, durch LOAD »*«,8. Nach dem Programmstart beginnt bereits eine recht eindrucksvolle Multitasking-Demo, bei der mehrere Operationen scheinbar gleichzeitig ablaufen. Die fünf Buchstaben-Sprites F,O,R,T und H verfolgen den frei verschiebbaren Cursor über den Bildschirm. Nachdem die Demo gelöscht wurde, befindet man sich im Forth-Interpreter, in dem alle fehlerfreien Eingaben sofort ausgeführt werden. Dieses Interpreter-/Compiler-Konzept ist ein wesentliches Merkmal des gesamten Forth-Konzeptes. Fast alle Eingaben können über den Interpreter-Modus sofort getestet und gegebenenfalls korrigiert werden. Dies geschieht, ohne den langwierigen Edit-Compile-Link-Zyklus anderer Compilersprachen gehen zu müssen. Über den ebenfalls integrierten Editor können Programme eingegeben, editiert und gespeichert werden. Beim Laden werden die Programme sofort compiliert, was zum Beispiel einen gewaltigen Geschwindigkeitsvorteil gegenüber Basic-Programmen bringt. So behält man die Flexibilität des Interpreter-Modus sowie die hohe Geschwindigkeit compilierter Forth-Programme. Eingegeben werden die Programme wie in den meisten anderen Forth-Umsetzungen auch bei Ultraforth 83 über »Screens«. Das sind Textseiten, in denen das Programm erstellt wird und die dann hintereinander auf der Diskette abgelegt werden. Der Befehlsumfang von Ultraforth 83 ist zwar nicht so überdimensioniert wie bei Super-Forth, jedoch sehr umfangreich und reicht vollkommen zur Programmierung aus. Der Anwender wird dadurch erst richtig angeregt, eigene Befehlsworte zu definieren, worauf die Philosophie von Forth gerade abzielt. Hervorragendes wurde von den Programmierern auch bei der Umsetzung des Multitasking auf den C64 geleistet, so daß es sogar Druckerspooler gibt. Dies sind Programme, die im Hintergrund ablaufen und Text oder Grafik ausdrucken, ohne das Hauptprogramm aufzuhalten. Weiterhin sind in der C64-Version leistungsfähige Grafikbefehle als eigene Forth-Wörter vordefiniert. Sie können in eigenen Programmen ebenso verwendet werden wie alle bereits zur »Grundausstattung« eines jeden Forth-Systems gehörenden Worte.

Starke Befehle

Das ganze System hält sich übrigens an den im Jahr 1983 definierten 83'er Standard, der bisher auf Computern wie dem C64 kaum bis gar nicht anzutreffen war. Es handelt sich dabei um eine leistungsfähige Weiterentwicklung des ursprünglich auch als Public Domain Version verbreiteten fig-Forth. Ultraforth 83 wurde aber auch um einige für die Programmierung wichtige Befehle erweitert. Diese sind zwar nicht im Standard beschrieben, jedoch durchaus sehr nützlich. Zum Beispiel die vielen C64-spezifischen Befehle, wie »curoff«, der ein Ausschalten des Cursors bewirkt, etc.

Neben den weiter oben schon angesprochenen Grafikbefehlen wie Grafik ein/aus (graphic/nographic) enthält das System auch eine vollständige Implementation der »Turtle-Grafik«-Befehle (pencolor, right, left etc.) sowie starke SPRITE-Routinen. Die unter Forth übliche UPN-Schreibweise bleibt natürlich auch hierbei erhalten. UPN bedeutet Umgekehrte Polnische Notation, auch Präfix-Schreibweise genannt, und beschreibt eine spezielle Eingabeart mathematischer Ausdrücke. Dabei werden zuerst die Zahlen und dann die auszuführenden Rechenzeichen eingegeben. Diese Schreibweise wird auch bei verschiedenen programmierbaren Taschenrechnern verwendet und ist zwar etwas gewöhnungsbedürftig aber auch ungeheuer flexibel. Sie kann, als interessantester Effekt, mit sehr hoher Geschwindigkeit interpretiert und ausgeführt werden. Näheres dazu im Beispiel 2 am Ende dieses Artikels.

Handbuch für Könner

Das Handbuch zu Forth gliedert sich in vier Teile sowie einen Anhang, der sich hauptsächlich den auf den Disketten be-

findlichen Programmen und Hilfsmitteln sowie den Grafikbefehlen widmet. Außerdem werden in diesem Teil auch Abweichungen von Ultraforth 83-Befehlen zu den in verschiedenen Lehrbüchern verwendeten Ausdrücken sowie die Fehlermeldungen angesprochen. Der erste Teil umfaßt Erläuterungen zu den verschiedenen systemspezifischen Forth-Strukturen sowie zum allgemeinen Aufbau von Forth-Worten und deren Verknüpfung im Speicher. Weiterhin werden die Multitasking-Fähigkeiten, Möglichkeiten zur Fehlersuche und die Anpassung von Ultraforth 83 an andere Computer beschrieben. Im zweiten Teil werden sämtliche Befehle, die irgendwie mit Ultraforth 83 zusammenhängen, genau aufgeführt und in einem kurzen Erklärungstext beschrieben. Eine Begriffsdefinition findet sich in Teil drei. In Teil vier schließlich wird der sehr gute Bildschirmeditor für den C64 recht ausführlich beschrieben.

Abschließend läßt sich sagen: Ultraforth kann im Vergleich zu anderen Forth-Varianten spielend mithalten. Zusätzlich hat es aber den großen Vorteil, daß man in den Besitz all seiner Fähigkeiten fast kostenlos gelangen kann. Besonders die hervorragenden Befehle zum Multitasking-Betrieb, die Grafikbefehle und der gute Bildschirmeditor verdienen volles Lob. Sie beweisen, daß auch und gerade Freesoft-Programme sehr gute Leistungsmerkmale aufweisen können. Ultraforth 83 ist also eine rundum gelungene Umsetzung dieser interessanten Programmiersprache auf dem C64.

Als Beispiele zeigen wir Ihnen noch zwei kleine Forth-Programme, die sehr gut die einfache Struktur einer Schleife sowie die UPN-Schreibweise verdeutlichen.

Beispiel 1.

```
: L1 DO 10000 1 I . LOOP ;
```

Dieses kleine Forth-Programm durchläuft eine Schleife 10000 mal und gibt jeweils den Schleifenindex auf dem Bildschirm aus. Der Doppelpunkt am Anfang ist ein festes Forth-Wort, das den Beginn einer neuen Wortdefinition festlegt. »L1« ist der Name des neuen Wortes, unter dem das »Programm« nun immer aufgerufen werden kann. »DO« und »LOOP« bilden den Schleifenrahmen. Da die Schleife den Rahmen von 1 bis 10000 durchlaufen soll, wird zuerst 10000 auf den Stapel gelegt und danach die 1. »I« legt den Schleifenindex auf den Stapel. ».« ist mit dem »PRINT« von Basic vergleichbar. Es holt den obersten Stapelwert, zeigt ihn dann auf dem Bildschirm an und löscht ihn danach vom Stapel. »;« beendet schließlich die Wortdefinition.

Beispiel 2.

Die Rechnung 3*7+25 sähe in Präfix-Schreibweise so aus: 25 3 7 * +

Zuerst werden die Zahlen 25, 3 und 7 auf dem Stapel plaziert. Der Stapel sieht dann folgendermaßen aus (TOS=Top of Stack bedeutet oberstes Stapelelement):

```
 7 (TOS)
 3
25
```

Das Wort »*« multipliziert die zwei obersten Stapelelemente und speichert das Ergebnis anstelle dieser Zahlen als TOS:

```
 7
 3 ---> 21 (TOS)
25       25
```

Schließlich wird mit dem Befehl »+« eine Addition durchgeführt und das Ergebnis wieder als TOS gespeichert.

```
21
25 ---> 46 (TOS)
```

(Ingolf Krüger/jk)

Bezugsquelle: Forth-Gesellschaft eV, Schanzenstraße 27, 2000 Hamburg 6, Tel.: 040-435070 (Mo 11-18 Uhr)

Basic im Galopp

Wie schnell sind Basic-Compiler wirklich? Bringen sie den erhofften Geschwindigkeitsgewinn gegenüber interpretiertem Basic? Wir haben die Compiler für den C64 und C128 einem gründlichen Test unterzogen.

Compiler ist das Zauberwort, das jedem Basic-Programmierer einen Hauch von Maschinensprache vermittelt. Und je mehr davon, desto besser. Wie weit die Compiler an die Maschinensprache heranreichen und welche Vorzüge sie sonst noch bieten, haben wir zum Gegenstand dieses Tests gemacht.

Doch bevor wir uns die Compiler näher ansehen, sollten wir uns mit den Grundlagen der Compilierung vertraut machen. Im Vordergrund steht beim Compiler natürlich die Geschwindigkeit. Dann sollte das Basic-Programm am besten ohne Änderungen compilierbar sein. Das kann aber schon zu Problemen führen. Denn eventuell hat man eine Befehlserweiterung geladen, die nicht übersetzbar ist. Oder man muß um der Geschwindigkeit willen den Compiler durch spezielle Befehle steuern. Zum Beispiel bei Schleifen, die von Integer-Variablen abhängig sind. Im Basic 2.0 ist das verboten, bei Compilern läßt sich hier viel Zeit gewinnen.

Der Grundgedanke ist einfach: aus Basic soll Maschinensprache werden. Doch leider finden nicht alle Basic-Befehle ihre Entsprechungen in der Maschinensprache. Eher das Gegenteil ist der Fall. Alle Berechnungen, die Dezimalzahlen enthalten, String-Operationen, IF-THEN-Abfragen etc. müssen in ganze Befehlsketten zerlegt werden. Sprünge, Unterprogrammaufrufe, POKEs, DATA-Zeilen oder Ausgaben mit PRINT sind dagegen noch einfach zu übersetzen. Die Übersetzung in reine Maschinensprache wäre immer mit erheblichen Speicherplatzverlusten verbunden.

Eine zweite Möglichkeit ist die Erzeugung des sogenannten »Adreßcodes«. Dazu wird das Programm nicht in Assembler-Code übersetzt, sondern als Adreß-, Speed- oder Pseudo-Code (P-Code). Also nicht in Maschinensprache, sondern eine dem Basic ähnliche Struktur. Dieser P-Code ist extrem kurz, er benötigt im günstigsten Fall knapp die Hälfte eines Basic-Programms. Zur Ausführung benötigt er allerdings noch einen speziellen Interpreter, eine Programmbibliothek und etwas mehr Zeit. Der neue Interpreter und die Programmbibliothek bilden das »Run-Time-Modul«. In den meisten Fällen ist die Ausführungszeit nur wenig höher als bei echtem übersetzten Maschinen-Code. Dieser ist nämlich nur dann effektiver, wenn das Basic-Programm sich mit wenigen Maschinensprachebefehlen übersetzen läßt. Bei transzendenten Funktionen, wie Sinus- oder Wurzelberechnung, muß auch der Maschinen-Code auf die langsameren Gleitkomma-Routinen zurückgreifen.

Zum Test angetreten sind fünf Compiler, drei für den C64, zwei für den C128. Dies sind Austro-Comp, Austro-Speed, Basic 64, Austro-Comp 128 und Basic 128. In Tabelle 1 fin-

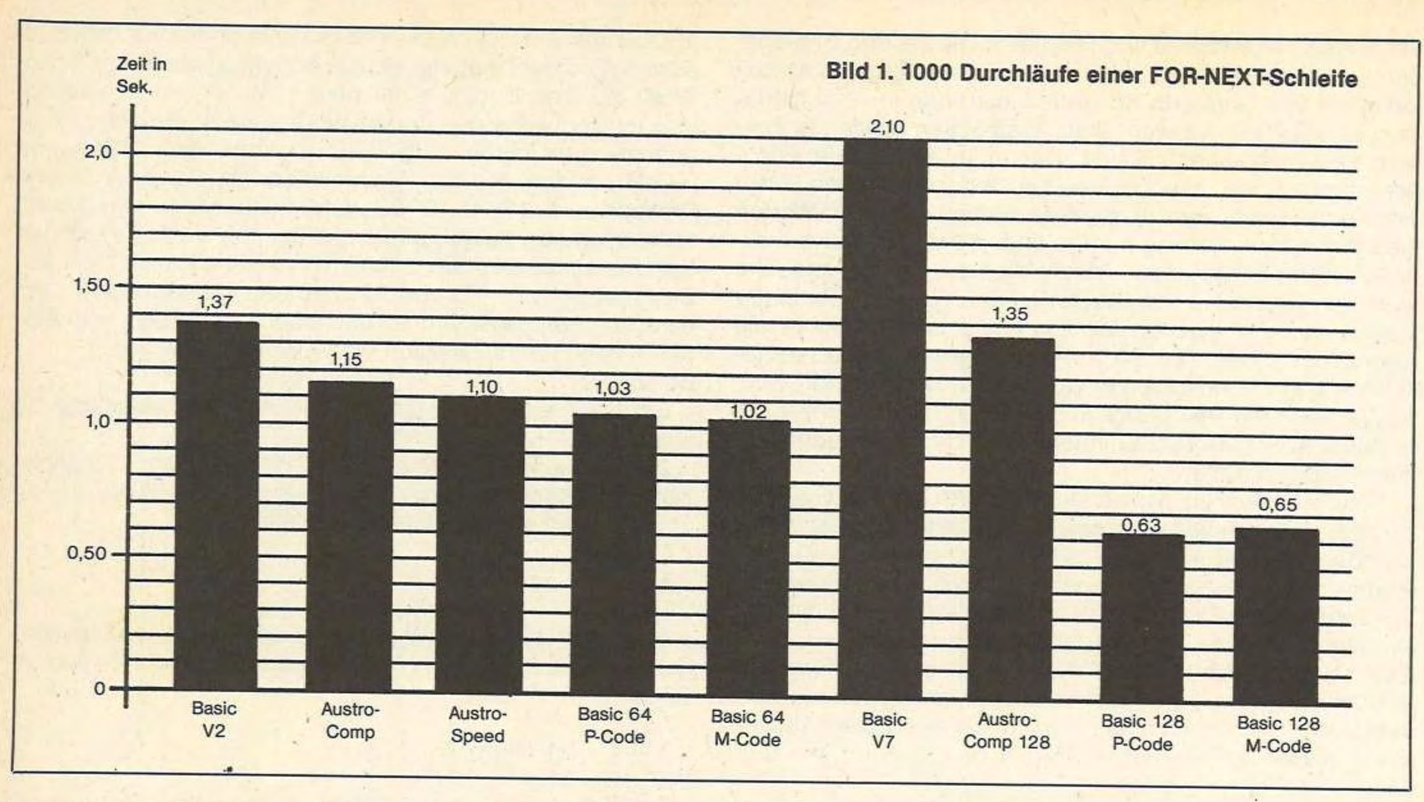


Bild 1. 1000 Durchläufe einer FOR-NEXT-Schleife

Bild 2. 1000 IF-THEN-Abfragen. Erstaunlich ist, daß im M-Code des Basic 128 die IF-THEN-Abfrage schneller ist als die FOR-NEXT-Schleife.

Zeit in Sek.
15,0
11,0
10,0
5,0
0
14,40
10,20
2,07
1,38
1,85
1,23
1,93
0,95
0,43
Basic V2
Austro-Comp
Austro-Speed
Basic 64 P-Code
Basic 64 M-Code
Basic V7
Austro-Comp 128
Basic 128 P-Code
Basic 128 M-Code

den Sie die dazugehörigen Daten: Preis, Lieferumfang und Bezugsadresse.

Wie man einen Compiler testet? Man mißt ganz einfach die Zeit. Dabei hilft uns ein Programm, das die wichtigsten zeitkritischen Programmteile enthält. Jeder Teil wird im Test 1000 mal durchlaufen, so daß äußere Einflüsse (zum Beispiel Interrupts) nicht ins Gewicht fallen. Die interne Uhr TI$ ist für den Test ausreichend genau. Ein solcher Benchmark-Test ist in Listing 1 abgedruckt. Vor und nach jedem Schleifendurchlauf wird die Anfangs- beziehungsweise Endzeit gespeichert. Am Schluß des Programmes, ab Zeile 900, wird dann die Laufzeit ermittelt. Die Tests gliedern sich wie folgt:

Benchmark 1: FOR-NEXT-Schleife
Benchmark 2: IF-THEN-Schleife
Benchmark 3: Rechnen mit +,-,*,/,^ und Klammern
Benchmark 4: GOSUB und RETURN
Benchmark 5: Indizierte Felder beschreiben
Benchmark 6: Funktionen SIN, TAN, EXP, VAL, STR$

	Austro-Comp	Austro-Speed	Austro-Comp 128	Basic 64	Basic 128
Lieferumfang	1 Diskette	1 Diskette	1 Diskette	1 Diskette	1 Diskette
	1 Handbuch	1 Handbuch	1 Handbuch	1 Handbuch	1 Handbuch
	1 Dongle	1 Dongle	1 Dongle		
Preis	Bei A-Speed enthalt.	129 Mark o. MwSt.	190 Mark o. MwSt.	99 Mark	99 Mark
Bezugsadresse	Digimat	Digimat	Digimat	Data Becker	Data Becker

Tabelle 1. Bezugsadresse und Preise der Compiler. Ein Dongle ist ein Kopierschutzstecker.

	Basic V 2	Austro-Comp	Austro-Speed	Basic 64 P-Code	Basic 64 M-Code	Basic V7	Austro-Comp 128	Basic 128 P-Code	Basic 128 M-Code
Benchmark 1:	1,37	1,15	1,10	1,03	1,02	2,10	1,35	0,63	0,65
Benchmark 2:	10,20	2,07	1,38	1,85	1,23	14,40	1,93	0,95	0,43
Benchmark 3:	70,92	65,35	63,93	62,47	61,45	77,55	41,62	26,95	26,11
Benchmark 4:	5,32	1,35	1,25	1,35	1,30	13,52	1,57	0,97	0,95
Benchmark 5:	8,78	3,67	2,78	3,25	2,77	13,38	4,48	2,87	2,36
Benchmark 6:	130,07	122,42	113,97	120,35	119,35	142,57	126,57	65,50	64,70
Benchmark 7:	25,87	2,75	2,63	2,47	2,25	49,35	2,50	2,40	2,18
Benchmark 8:	144,16	74,95	61,60	40,53	33,52	181,38	52,18	36,90	30,78
Gesamtzeit:	100,0%	69,0%	62,7%	58,8%	56,2%	124,6%	58,5%	34,6%	32,3%
Compilierdauer für EDDI		2:50	3:00	6:20			2:20	4:00	
Länge des Compilates EDDI (Bl=Blocks)		24 Bl.	32 Bl.	32 Bl.	40 Bl.		51 Bl.	60 Bl.	

Tabelle 2. Die Ergebnisse der Benchmark-Tests im Überblick

Austro-Comp, Austro-Speed und Austro-Comp 128: Zeilennummern 100, 110, 120, 130, 150, 160 (Listing 2)
Basic 64 und Basic 128: Zeilennummern 40, 60, 100, 110, 120, 130, 150, 160 (Listing 2)

Tabelle 3. Welche Fehler wurden von den Compilern erkannt?

Benchmark 7: READ und RESTORE
Benchmark 8: Zufallsfeld mit 100 Elementen mit Bubble-Sort-Algorithmus sortieren.

Test 9: zweimal UNDEF'D STATEMENT (falsche Zeilennummer) und einmal SYNTAX-ERROR provozieren. Zeile 750 enthält eine im Basic V2 nicht erlaubte Schleifenvariable I%. In 770 wird gewartet, bis die <RETURN>-Taste gedrückt und wieder losgelassen wurde.

Die Ergebnisse des Benchmark-Tests sind in Tabelle 2 zusammengefaßt und in den Bildern 1 bis 8 grafisch dargestellt. Die Ergebnisse dieser Tests liefern allerdings kein optimiertes Bild jedes einzelnen Compilers. Durch Eingriffe in das Programm und Anweisungen an den Compiler können in der Regel noch höhere Leistungen erzielt werden. Für den C128 mußte die Adresse in Zeile 770 geändert werden. Es heißt dann jeweils »...WAIT 212,...«.

Der zweite Lauf ist ein Test der Compilierdauer. Wir greifen hier auf das schon im letzten Compiler-Test (64'er, Ausgabe 2/85) verwendete Programm »Eddi« zurück. Dabei gibt es bei den C128-Compilern kein lauffähiges Compilat, denn die Abfrage der Funktionstasten ist für den C64 ausgelegt und somit nicht auf den C128 übertragbar. Das ist auch nicht so wichtig, es kommt hier nur auf die Zeit an, die zum Compilieren benötigt wird. Eddi bietet dabei noch zusätzliche Tücken. Die sind in der absolut Basic-typischen Spaghetti-Programmierung enthalten, welche sogar einen Programmfehler hervorgerufen hat: Zeile 1070 von Eddi enthält eine IF-Abfrage, deren Sprungziel nicht existiert. Die Abfrage ist jedoch sinnlos, der Fehler kommt im Ablauf nicht zum Tragen. Alle fünf Compiler haben auf diese Situation richtig reagiert: Der Fehler wurde angezeigt, beeinflußte jedoch den Compilierablauf nicht.

Geschwindigkeit allein reicht aber nicht aus. Auch Programmierfehler wie in Zeile 1070 sollte ein Compiler ohne Absturz überwinden – zumindest darf er sie nicht ohne Meldung übergehen. Darum haben wir Listing 2 mit Fehlern vollgespickt. Lediglich Zeile 10 enthält keinen »echten« Fehler, sondern eine undimensionierte Feld-Variable.

Austro-Comp & Austro-Speed

In Basic sind Felder maximal bis zum Index 10 zulässig. Da ein Compiler auch diese Felder reservieren muß, wird zusätzlich die selbständige Dimensionierung überprüft. Es darf also keine Fehlermeldung erscheinen. Einige Fehler sind während des Compilierens kaum zu finden. Welche Fehler erkannt wurden, entnehmen Sie bitte Tabelle 3.

Austro-Comp und die verbesserte Version Austro-Speed werden mit einem ausreichenden Handbuch und einem besonderen Kopierschutz, dem Dongle, auf Diskette ausgeliefert. Ein Dongle ist ein Hardwarezusatz, der in den User-

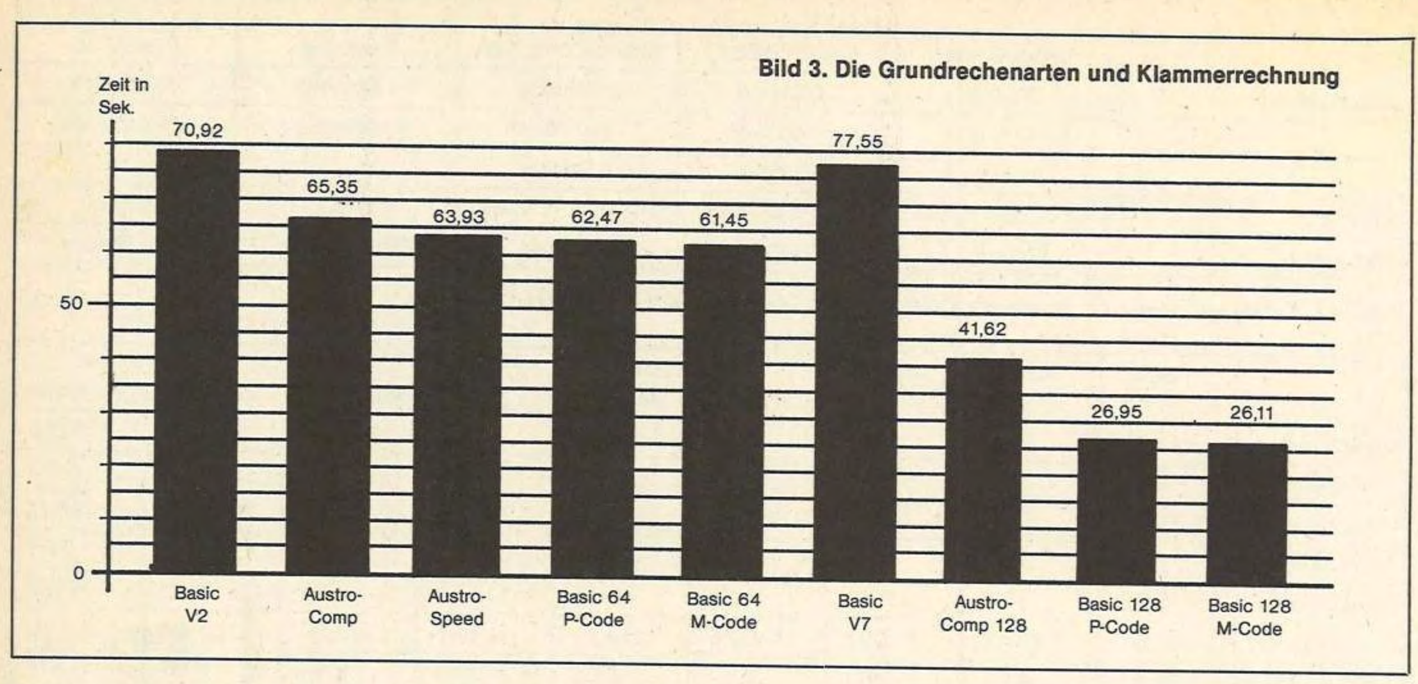


Bild 3. Die Grundrechenarten und Klammerrechnung

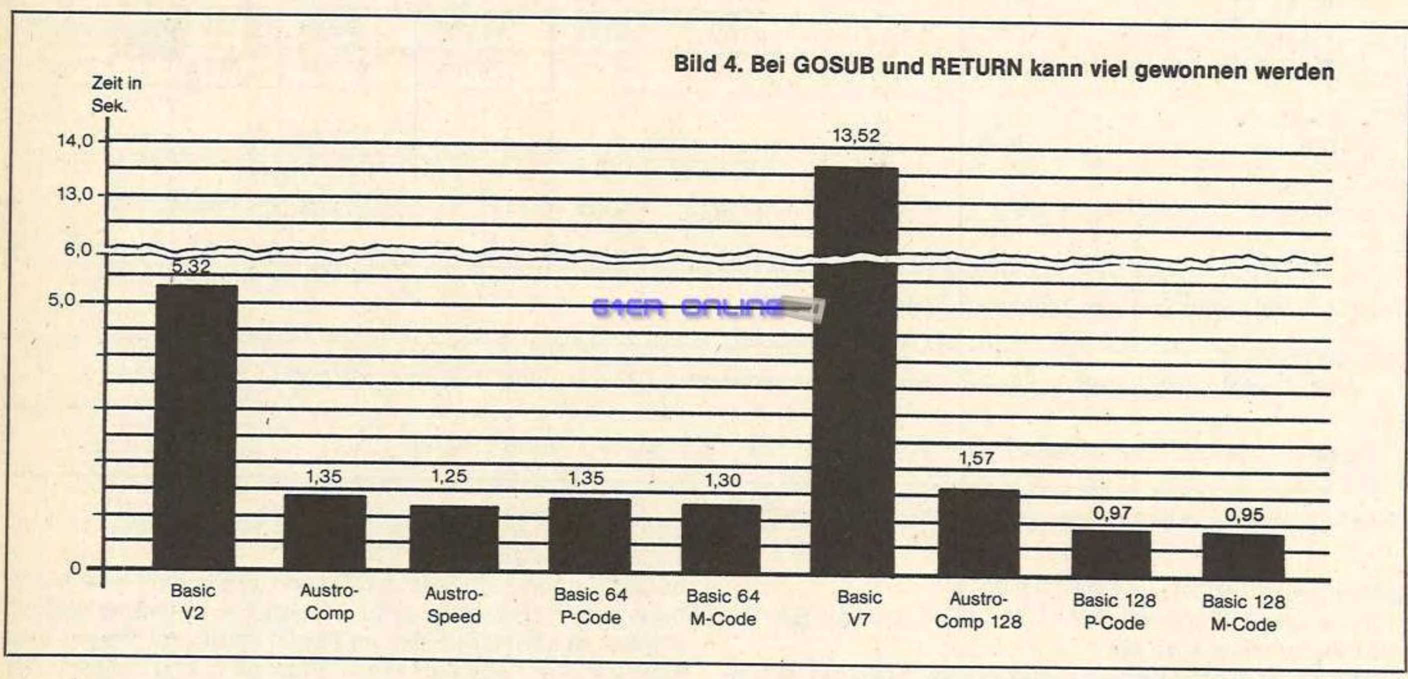


Bild 4. Bei GOSUB und RETURN kann viel gewonnen werden

Port eingesteckt wird. Das Programm fragt das Vorhandensein ab und läuft immer nur mit einem bestimmten Dongle. Die Bedienung beider Programme ist identisch.

Bevor man den Compiler startet, sollte man sich vergewissern, ob auf der Diskette genug Platz vorhanden ist, um das Compilat, ein Zeilen-File und noch weitere, vom Compiler wieder gelöschte Hilfs-Files aufzunehmen. Das Zeilen-File enthält die Speicheradressen aller Programmzeilen. Das ist wichtig für den Fall, daß ein Laufzeitfehler auftritt. Das compilierte Programm meldet dann nur die Adressen, so daß der Fehler anhand des Zeilen-Files lokalisiert werden kann.

Eine Besonderheit beider Compiler sind dynamische Arrays. Es ist im Gegensatz zu den meisten anderen Compilern erlaubt, Felder durch Variable zu dimensionieren. Die Anweisung DIM A(N) ist also erlaubt.

In zwei Durchläufen arbeiten die Austro-Compiler das Basic-Programm Eddi ab. Austro-Comp benötigt dafür 2:50 Minuten, Austro-Speed genau drei. Beide entdecken den Programmfehler und erzeugen ein 24 beziehungsweise 32 Blöcke langes Compilat und das Zeilen-File.

Basic 64 wird auf Diskette und mit einem Handbuch mittleren Umfangs ausgeliefert. Die Dokumentation ist durchweg gut, und führt in alle Funktionen des Compilers ein. Auch die Compiler-eigenen Zusätze (Compiler-Direktiven) sind im Handbuch ausreichend erklärt.

Basic 64

Basic 64 ist ein 2-Pass-Compiler, der jedoch über wesentlich mehr Einstellmöglichkeiten verfügt als die Austro-Compiler. Die Bedienung ist einfach, da sie durch ein Menü gesteuert wird. Dabei bietet er eine Fülle von Einstellmöglichkeiten und Compiler-Direktiven (Steuerung des Compilers aus dem Basic-Programm heraus). Basic 64 kann, wenn nötig, Maschinen-Code erzeugen und ihn mit dem P-Code vermischen. So können zum Beispiel zeitkritische Unterprogramme im Maschinen-Code compiliert sein, während das Hauptprogramm im platzsparenden P-Code geschrieben ist.



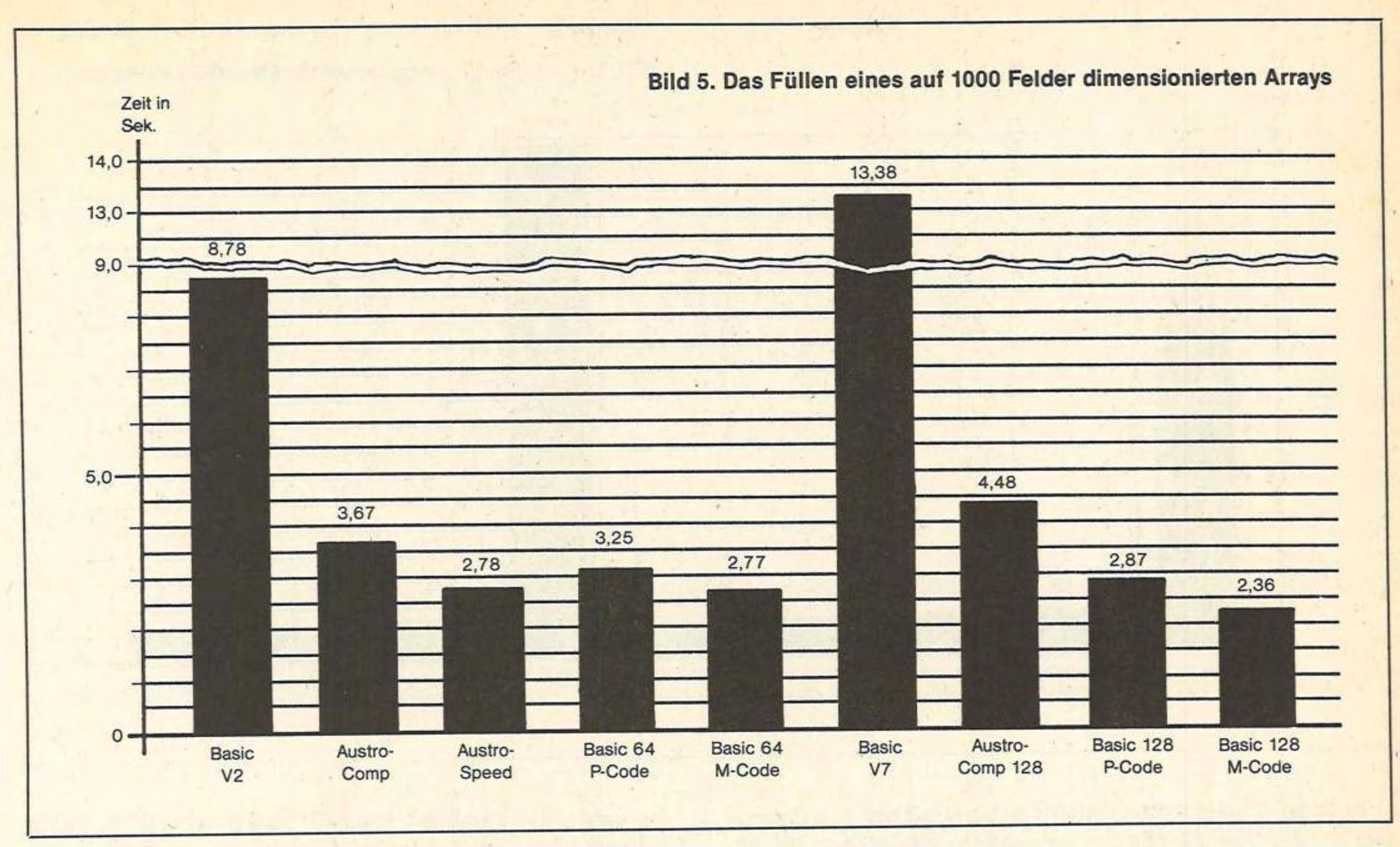


Bild 5. Das Füllen eines auf 1000 Felder dimensionierten Arrays

Bild 6. Transzendente Funktionen (SIN, TAN, EXP, sowie VAL und STR$) werden kaum beschleunigt, da die Routinen des Interpreters benutzt werden. Nur Basic 128 benutzt eigene Rechenroutinen.

Zeit in Sek. (Achse: 0, 50, 100, 150)

Basic V2	Austro-Comp	Austro-Speed	Basic 64 P-Code	Basic 64 M-Code	Basic V7	Austro-Comp 128	Basic 128 P-Code	Basic 128 M-Code
130,07	122,42	113,97	120,35	119,35	142,57	126,57	65,50	64,70

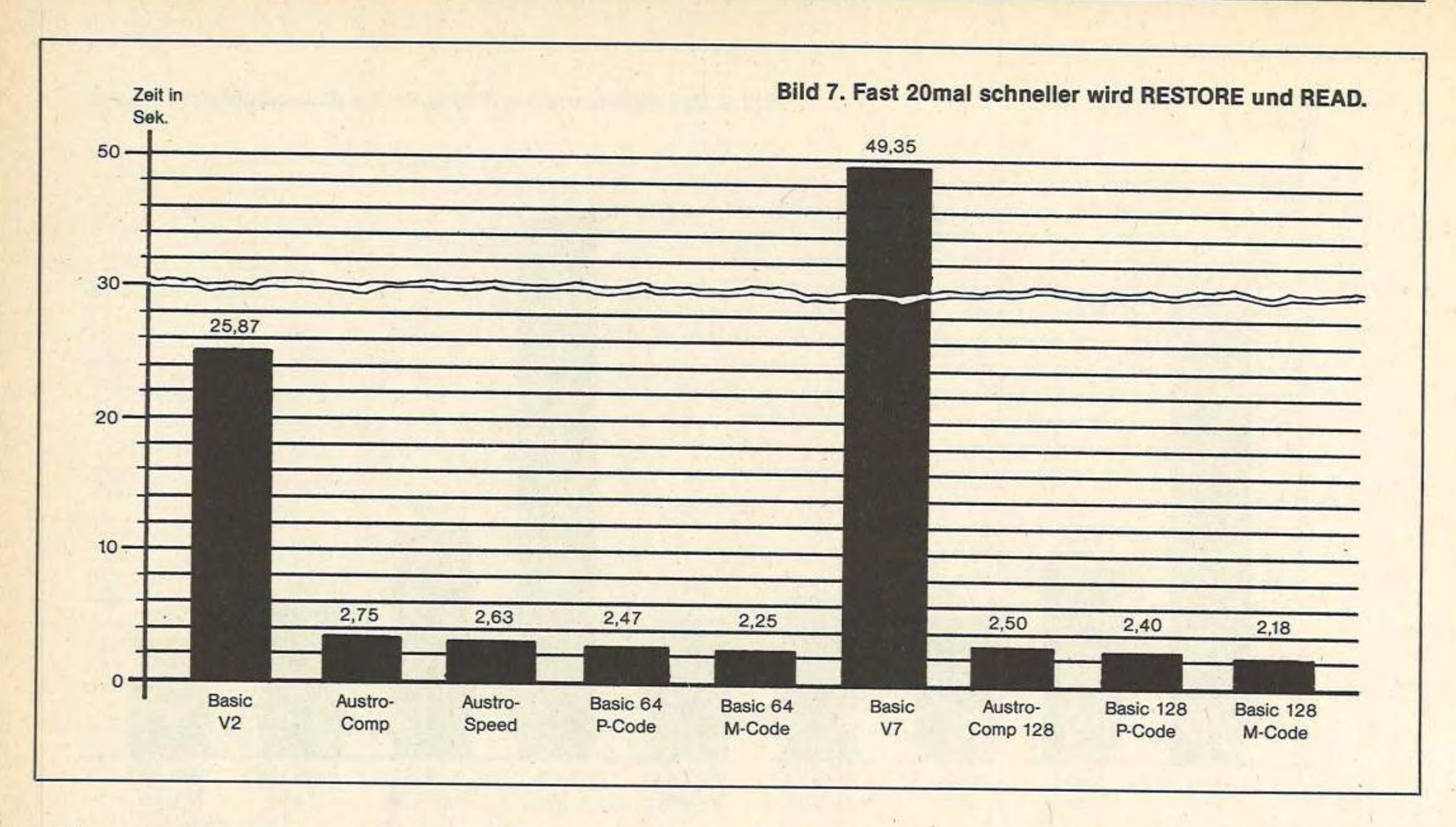


Bild 7. Fast 20mal schneller wird RESTORE und READ.

Weitere Möglichkeiten sind freies Verschieben des Compilats im Speicher, die Nutzung von mehr als 60 KByte, modifizierte Integer-Arithmetik und die Simulation von ON ERROR GOTO-Befehlen.

Eine zusätzliche Eigenschaft, die ihn von anderen Compilern abhebt, ist der Warm-Overlay: Beim Nachladen von Programmteilen werden die Variablen nicht gelöscht. Somit kommt Basic 64 dem Interpreter näher als andere.

Der einzig einschränkende Faktor besteht in der variablen Dimensionierung (DIM A$(N)). Dies ist durch die spezielle Variablenbehandlung der Compiler nicht zulässig. Tritt trotzdem eine solche Dimensionierung auf, fragt das Programm nach der Feldgröße, um diese für die Variable einzusetzen.

Auch vor etwaigen Erweiterungen macht Basic 64 nicht halt. Simons Basic, ExBasic Level II, Supergrafik 64, Supergrafik 64+ und Basic 4.0 werden in uncompilierten Code (Non-Compiled-Code) abgelegt und zur Ausführung an den Interpreter übergeben.

Nach 6:20 Minuten ist Basic 64 mit Eddi fertig. Auf der Diskette befindet sich nun das 32 (40) Blocks lange P-(M-) Compilat. Der Fehler in Zeile 1070 wurde erkannt, es wurde trotzdem ein lauffähiges Compilat erstellt.

Austro-Comp 128

Der C128-Compiler wird wie seine C64-Kollegen auf Diskette, mit Handbuch und Dongle ausgeliefert. Bis auf vier Befehle ist Austro-Comp 128 zum Basic 7.0 kompatibel. Diese Einschränkungen betreffen die Befehle TRAP, RESUME, COLLISION und GRAPHIC CLR. Bei TRAP, RESUME und COLLISION dürfen keine Variablen angewandt werden. Statt GRAPHIC CLR wird lediglich GRAPHIC 0 ausgeführt.

In zwei, für Overlay-Files drei Durchläufen bearbeitet Austro-Comp 128 das auf der Diskette befindliche Basic-Programm. Fünf Direktiven können den Compiler währenddessen steuern. Eine davon ist die Überprüfung des Dongles. Somit kann auch das Compilat geschützt werden.

Bei der Arbeit ist es dem Compiler möglich, auf zwei Laufwerke oder ein Doppellaufwerk zuzugreifen.

Die erstellten Overlay-Files sind Kalt-Overlays, da keine Variablen oder deren Inhalte übernommen werden. Dafür muß nur ein Programm das etwa 11 KByte lange Run-Time-Modul enthalten. Bei Programmpaketen, bei denen das erste Programm als Menü dient, die Folgeprogramme aber unabhängig sind, lohnt sich das Kalt-Overlay.

Variable Dimensionierung wird vom Austro-Comp 128 unterstützt. Dies wird möglich durch ein spezielles Zeigerarray »z*%«, das immer dann angelegt wird, wenn im Source-File Arrays vorkommen. Der Array-Name z*% ist im Basic unmöglich, damit sind Kollisionen der Variablen ausgeschlossen.

Basic 128

Trotz des im SLOW-Modus langsameren Basic V7.0 ist das Compilat der Benchmark-Tests schneller als auf dem C64. Das Testprogramm Eddi wurde in 2:20 Minuten abgearbeitet. Mit 51 Blocks ist das Compilat ziemlich lang, was jedoch einleuchtend ist, da im Run-Time-Modul der Befehlssatz des C128 als Ballast mitgeschleppt wird. Erstaunlicherweise war das Programm fast lauffähig. Lediglich die Funktionstastenabfrage entsprach nicht dem C128.

Bestehend aus Diskette und einem sehr ausführlichen Handbuch wird Basic 128 ausgeliefert. Die speziellen Optimierungsverfahren für Source-Programme sind beispielhaft erklärt, beim Lesen wird man in die Arbeitsweise des Compilers hineinversetzt.

Basic 128 ist eine Weiterentwicklung des Basic 64. Er ist kompatibel zum Basic V7.0, viele Funktionen sind verbessert. So sind eigene Rechen-Routinen enthalten, Warm-Overlays mit Variablen-Inhalten sind durchführbar, die im Basic 7.0 undokumentierten Befehle werden unterstützt.

Neben den üblichen Compiler-Optimierungen führt das Run-Time-Modul Laufzeit-Optimierungen der Variablen durch. Da Integer-Variablen nicht in jedem Programm gekennzeichnet sind, ist die Laufzeit-Optimierung bei nicht an den Compiler angepaßten Programmen durchaus vorteilhaft.

Zeit in Sek.
180
150
140
80
50
0
144,16 74,95 61,60 40,53 33,52 181,38 52,18 36,90 30,78
Basic V2 | Austro-Comp | Austro-Speed | Basic 64 P-Code | Basic 64 M-Code | Basic V7 | Austro-Comp 128 | Basic 128 P-Code | Basic 128 M-Code

Bild 8. Sortieren von 100 Zufallselementen mit Bubble-Sort

Zur optimalen Programmierung stehen 15 Compiler-Direktiven bereit. Zur Verkettung oder zum Overlay von Programmen und Programmteilen sind Symboltabellen ladbar. Dadurch verwenden dann die Programme gleiche Variablen-Adressen.

Eddi wurde in 4:00 Minuten zu 60 Blocks P-Code compiliert (M-Code war bei dem C 64-Programm nicht möglich). Es funktionierte auch hier wieder bis auf die Funktionstastenabfrage. Die Benchmark-Tests verliefen in einer erstaunlichen Geschwindigkeit, obwohl keine Compiler-Direktiven und kein Fast-Modus benutzt wurde. Die von Basic 128 erzeugten Programme sind um fast ein Drittel schneller als die der anderen Compiler.

Superschnell

Für den C 128 haben wir uns noch etwas Besonderes ausgedacht. Denn im FAST-Modus sind die Programme doppelt so schnell wie Normal. Bei den Compilern Austro-Comp 128 und Basic 128 benutzten wir noch zusätzlich Compiler-Anweisungen, um die Programme auf die kürzestmögliche Ausführungszeit zu beschleunigen.

Wir gaben also in unserem Benchmark-Testprogramm noch in der ersten Zeile den FAST-Befehl und definierten alle Laufvariablen in Integer-Variablen um. Jetzt sieht das Ganze etwas anders aus, nämlich bis zu 170mal schneller (IF-THEN bei Basic 128). Auch die sonst so zeitintensiven Rechen- und Stringverarbeitungsroutinen (Benchmarks 3, 6 und 8) werden jetzt in erträglichen Zeiten abgearbeitet. Austro-Comp 128 benötigt insgesamt nur noch 20 Prozent, Basic 128 knapp 12 Prozent gegenüber Basic V7. Optimieren lohnt sich also bei Compilern sehr. Die Ergebnisse dieses Durchlaufs können Sie Tabelle 4 entnehmen.

	Basic V7		Austro-Comp 128	Basic 128
	SLOW	FAST		
Benchmark 1:	2,10	1,05	0,25	0,15
Benchmark 2:	14,40	7,20	0,57	0,06
Benchmark 3:	77,55	38,78	18,98	11,40
Benchmark 4:	13,52	6,75	0,35	0,31
Benchmark 5:	13,38	6,69	1,57	0,90
Benchmark 6:	142,57	71,29	56,62	31,05
Benchmark 7:	49,35	24,68	0,82	0,90
Benchmark 8:	181,38	90,69	18,91	12,70

Tabelle 4. Im FAST-Modus und durch die Verwendung von Compiler-Direktiven läßt sich beim C128 noch viel herausholen

Empfohlen wird...

Geschwindigkeit ist Trumpf. Und den hält eindeutig Basic 128 in der Hand. Für den C 128 ist, wenn Sie nicht unbedingt die variable Array-Dimensionierung benötigen, Basic 128 zu empfehlen. Seine Fähigkeiten nutzen den C 128 weitgehend aus.

Beim C 64 ist es eine Frage der Anwendung. Je nach Anforderung des Programms variiert die Geschwindigkeit des erzeugten Codes (siehe Benchmarks). Auch in der Bedienung ergeben sich keine einschneidenden Unterschiede. Dadurch wäre es nicht richtig, eine Entscheidung zu fällen, ohne die Anforderungen des Benutzers zu kennen.

(og)

Info:
Digimat, Arbeitergase 48, A-1050 Wien
Data Becker, Merowingerstraße 30, 4000 Düsseldorf

```
100 REM ********************************      <238>
110 REM BENCHMARKS FUER COMPILER, C 64        <066>
120 REM ********************************      <002>
130 REM                                        <192>
140 REM ************** 1 ** FOR NEXT ***      <011>
150 PRINT"{CLR,DOWN}AUSFUEHRUNG: 1{LEFT}";
    :A1=TI                                     <093>
160 FOR I=1 TO 1000                            <082>
170 NEXT I                                     <254>
180 E1=TI                                      <089>
190 REM ************** 2 ** IF THEN ****      <039>
200 PRINT"2{LEFT}";:A2=TI                      <193>
210 I=0                                        <143>
220 I=I+1                                      <203>
230 IF I<1000 THEN 220                         <190>
240 E2=TI                                      <157>
250 REM ************** 3 ** RECHNEN ****      <131>
260 PRINT"3{LEFT}";:A3=TI                      <019>
270 FOR I=1 TO 1000                            <194>
280 C=I+I*(I-I/I)↑(I/500)                      <015>
290 NEXT I                                     <120>
300 E3=TI                                      <227>
310 REM ************** 4 ** GOSUB ******      <230>
320 PRINT"4{LEFT}";:A4=TI                      <099>
330 FOR I=1 TO 1000                            <254>
340 GOSUB 800                                  <086>
350 NEXT I                                     <180>
360 E4=TI                                      <039>
370 REM ************** 5 ** FELDER *****      <255>
380 PRINT"5{LEFT}";:A5=TI                      <179>
390 DIM F(1000),F$(1000)                       <227>
400 FOR I=1 TO 1000                            <068>
410 F(I)=I:F$(I)="TEXT"                        <015>
420 NEXT I                                     <250>
430 E5=TI                                      <117>
440 REM ************** 6 ** FUNKTIONEN *      <189>
450 PRINT"6{LEFT}";:A6=TI                      <013>
460 FOR I=1 TO 1000                            <128>
470 C=SIN(I):C=TAN(I):C=EXP(I/30):C=VAL(ST
    R$(I))                                     <050>
480 NEXT I                                     <054>
490 E6=TI                                      <185>
500 REM ************** 7 ** READ DATA **      <093>
510 PRINT"7{LEFT}";:A7=TI                      <093>
520 FOR I=1 TO 1000                            <190>
530 RESTORE:DATA 0                             <087>
540 READ D                                     <096>
550 NEXT I                                     <126>
560 E7=TI                                      <009>
570 REM ************** 8 ** SORTIEREN **      <108>
580 PRINT"8{LEFT}";:A8=TI                      <185>
590 DIM S$(100)                                <051>
600 FOR I=1 TO 100                             <078>
610 FOR K=1 TO 10                              <008>
620 S$(I)=S$(I)+CHR$(RND(TI)*26+65)            <068>
630 NEXT K                                     <222>
640 NEXT I                                     <216>
650 FOR I=1 TO 99                              <045>
660 FOR K=I+1 TO 100                           <213>
670 IF S$(K)>S$(I)THEN 690                     <149>
680 S$(0)=S$(K):S$(K)=S$(I):S$(I)=S$(0)        <158>
690 NEXT K,I                                   <005>
700 E8=TI                                      <157>
710 REM ************* 9 ** FEHLER ? ***       <192>
720 GOTO 770                                   <044>
730 GOTO 729                                   <007>
740 GOSUB 739                                  <049>
750 FOR I%=1 TO 2:NEXT I%                      <212>
760 SYNTAX ERR OR                              <180>
770 PRINT"WARTEN":WAIT 197,1:WAIT 197,1,25
    5                                          <192>
780 GOTO 900                                   <010>
800 RETURN                                     <032>
900 REM ************* AUSWERTUNG ******       <070>
910 PRINT"{CLR,3DOWN}B1:"(E1-A1)/60"SEK.       <251>
920 PRINT"B2:"(E2-A2)/60"SEK.                  <111>
930 PRINT"B3:"(E3-A3)/60"SEK.                  <189>
940 PRINT"B4:"(E4-A4)/60"SEK.                  <011>
950 PRINT"B5:"(E5-A5)/60"SEK.                  <089>
960 PRINT"B6:"(E6-A6)/60"SEK.                  <167>
970 PRINT"B7:"(E7-A7)/60"SEK.                  <245>
980 PRINT"B8:"(E8-A8)/60"SEK.                  <069>
```

Listing 1. Die Benchmark-Testprogramme

```
10 A(9)=500                                    <222>
20 POKE 2,500:REM ILLEGAL QUANTITY             <220>
30 POKE 70000,5:REM ILLEGAL QUANTITY           <066>
40 PRINT CHR$(500):REM ILLEGAL QUANTITY        <226>
50 DIM X(70000):REM ILLEGALQUANTITY            <106>
60 PRINT 1/0:REM DIVISION BY ZERO              <198>
70 DIM A(100):DIM A(200):REM REDIM'D AR.       <076>
80 NEXT I:REM NEXT WITHOUT FOR                 <142>
90 RETURN:REM RETURN WITHOUT GOSUB             <232>
100 A=1E100:REM OVERFLOW                       <132>
110 SYNTAX:REM SYNTAX                          <201>
120 GOTO 0:REM UNDEF'D STATEMENT               <093>
130 GOSUB 0:REM UNDEF'D STATEMENT              <119>
140 READ A:REM OUT IF DATA                     <212>
150 A$=5:REM TYPE MISMATCH                     <002>
160 A="TEXT":REM TYPE MISMATCH                 <173>
170 PRINT FN A(8):REM UNDEF'D FUNCTION         <196>
180 PRINT#1:REM FILE NOT OPEN                  <035>
```

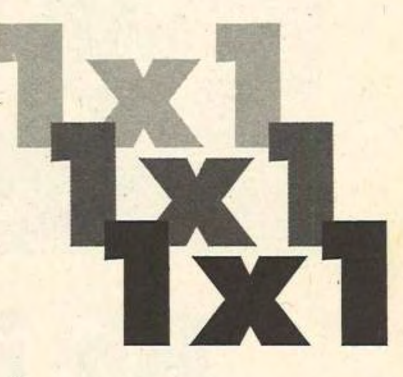
Listing 2. Welche Fehler werden erkannt?







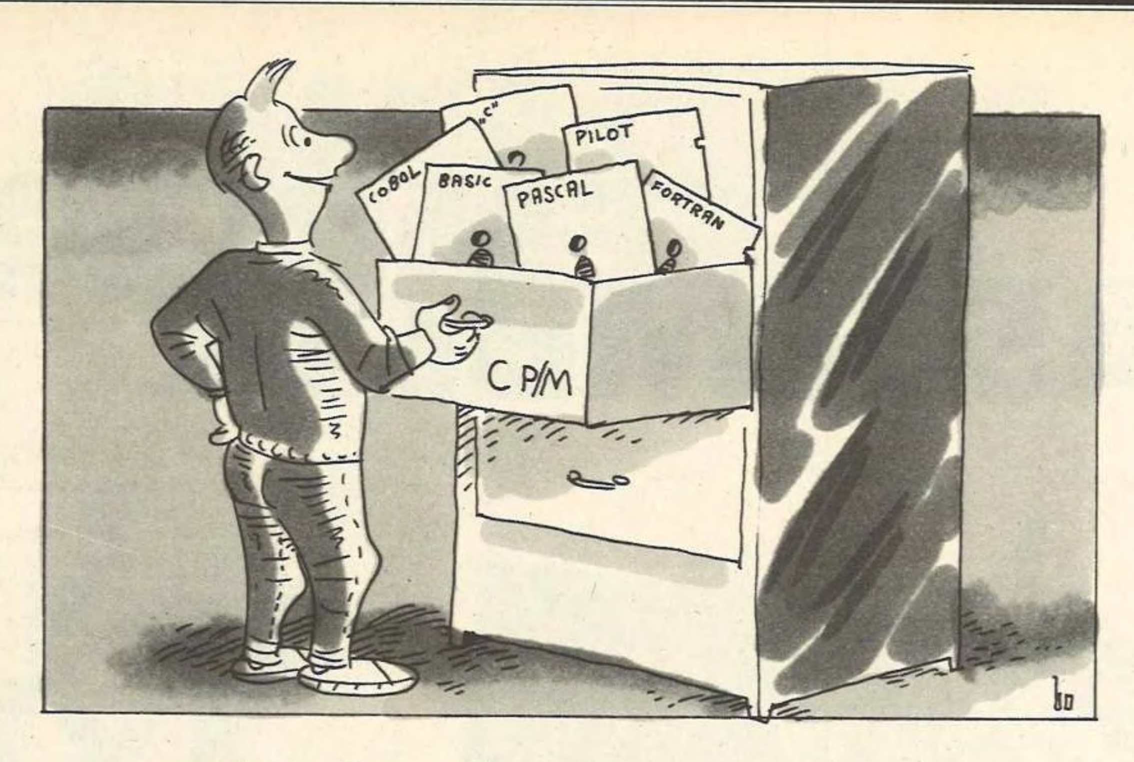
ASSEMBLER
ASSEMBLER
ASSEMBLER





Mit dem »64'er«-Sonderheft 8/85
Zwei umfassende Kurse vermitteln Programmiertechniken für Anfänger und Fortgeschrittene. Programme, Beispiele, wichtige Hinweise, viele Tips und Tricks erleichtern den Einstieg. Alle Listings haben natürlich ihre dokumentierten Quellcodes. Knifflige Aufgaben in Maschinensprache lassen sich optimal mit Hypra-Ass, SMON und einem zum Hypra-Ass kompatiblen Reassembler lösen. Als zusätzliches »Bonbon« sind im ersten Assembler-Sonderheft die wichtigsten Tabellen zusammengefaßt.

Nutzen Sie die Bestellmöglichkeit des Assembler-Sonderheftes 8/85 mit der eingehefteten Zahlkarte in diesem Sonderheft von »64'er«!



Für jeden etwas: Programmiersprachen unter CP/M

Eine Riesenauswahl an Programmiersprachen bietet der CP/M-Modus des C128. Wir berichten ausführlich über Leistungsmerkmale und informieren Sie über bisher vielleicht noch unbekannte Sprachen.

Schmerzlich vermißt der Programmierer unter CP/M das im C128 automatisch vorhandene Basic. Allerdings gibt es für CP/M eine Menge Compiler, deren Spektrum von Cobol bis C reicht. Jeder Programmierer wird hier seine Sprache finden und darüber hinaus feststellen, daß Komfort nicht unbedingt teuer bezahlt werden muß.

Die wohl am weitesten verbreitete Programmiersprache für Heimcomputer ist nach wie vor Basic. Eine ganze Reihe von Interpretern und Compilern stehen für CP/M zur Verfügung. Ein für 89 Mark (Handbuch in Englisch 29 Mark) sehr preiswerter und leistungsfähiger Interpreter ist Nevada-Basic. Dieser Interpreter überzeugt nicht nur durch sehr umfangreichen Sprachumfang (Bild 1), sondern auch durch die bedienerfreundliche Programmierumgebung.

Wie fast alle CP/M-Programme wird Nevada-Basic uninstalliert geliefert und muß daher erst an den C128 angepaßt werden. Dies geschieht mit dem mitgelieferten Programm »CONFIG«. Hier brauchen Sie nur das Terminal »Lear Seigler ADM 31« anzuwählen und schon ist die Installation fertig. Nach dem Laden von Nevada-Basic erscheint am Bildschirm die Einschaltmeldung mit der Anzeige des freien Speichers. Nun kann sofort mit der Eingabe des Programms begonnen werden. Zur Bearbeitung des Basic-Quellcodes stehen einige leistungfähige Befehle zur Verfügung. So sind die jedem Basic-Programmierer bekannten Anweisungen wie LIST und RUN voll implementiert. Das Inhaltsverzeichnis der Diskette kann mit CAT auf den Bildschirm geholt werden. Neue Zeilen werden einfach im Direktmodus eingegeben. Wollen Sie eine bestehende Zeile ändern, stoßen Sie auf eine Besonderheit des Interpreters. Mit dem aus dem C64- oder C128-Modus gewohnten LIST-Befehl wird zwar die gewünschte Zeile angezeigt, aber bei Betätigung der Cursor-Tasten tut sich gar nichts. Dafür bietet Nevada-Basic die Anweisung EDIT an. Das Programm wird dadurch ab der gewünschten Zeile gelistet, der Cursor auf der angegebenen Zeile positioniert. Nun steht für die Bearbeitung des Programms ein leistungsfähiger Bildschirm-Editor zur Verfügung, in dem der Cursor innerhalb der angezeigten Zeilen frei gesteuert werden kann (Full-Screen-Editor). Die Cursor-Steuerung erfolgt über die Cursor-Tasten oder über entsprechende Control-Sequenzen. Wird nach der Änderung einer Zeile <RETURN> gedrückt, schiebt der Editor automatisch eine REM-Zeile ein. Zum Speichern des erstellten Programms wird wie gewohnt der SAVE-Befehl verwendet. Geladen werden Programme mit GET.

Taucht während des Programmablaufs ein Fehler auf, so wird dieser am Bildschirm angezeigt. Durch Drücken von <RETURN> springt man dann in den Editor. Dort ist der Cursor sofort auf der fehlerhaften Zeile positioniert. Ein Aspekt, der etwas an den Editor von Turbo-Pascal erinnert. Natürlich können auftretende Fehler auch abgefangen werden. An den Anfang des Programms wird dazu einfach die Anweisung »ERRSET Zeilennummer« gestellt. Die Funktion

ist ähnlich dem Basic 7.0 des C 128, dort lautet der entsprechende Befehl TRAP. Hier kann nun die Basic-Fehlermeldung ausgegeben und entsprechend reagiert werden.

Sehr gut gelöst wurde auch die Listingausgabe. Das Programm wird formatiert am Bildschirm angezeigt oder auf Drucker ausgegeben. So wird der Inhalt einer FOR..NEXT-Schleife mit Einrückungen ausgegeben (Bild 2). Für die Ausgabe auf Drucker entfällt das Eröffnen von Kanälen oder die umständliche Kopiererei mit PIP. Es genügt der Befehl LLIST.

Die implementierten Befehle repräsentieren den nahezu klassischen Basic-Standard. Was der Basic 7.0-Programmierer schmerzlich vermissen wird, sind Strukturanweisungen wie etwa WHILE..DO. Dafür stehen aber mächtige Befehle zur Dateiverwaltung zur Verfügung, sowie eine umfangreiche Bibliothek an mathematischen Funktionen. Die Bearbeitung von sequentiellen und relativen Dateien bereitet keinerlei Probleme. Bei der Eröffnung einer Datei kann angegeben werden ob sie nur gelesen, nur beschrieben oder in beiden Modi bearbeitet werden soll. Der Befehl REWIND stellt Dateizeiger für sequentielle Dateien auf den ersten Satz zurück. Dies ist vor allem dann nützlich, wenn auf ein- und dieselbe Datei mehrmals hintereinander von Anfang an zugegriffen werden soll. Man erspart sich das ständige Öffnen und Schließen.

Vor allem für die Dateiverwaltung ist formatierte Ein-/Ausgabe wichtig. Bei Nevada-Basic kann hierzu der PRINT- wie auch der INPUT-Befehl entsprechend erweitert werden. Bei PRINT können Sie genau definieren, in welcher Form Sie Zahlen ausgeben wollen. So ist es möglich anzugeben, ob die Ausgabe der Zahl als Integer, Fließkommazahl, mit Vorzeichen oder mit vorgestelltem Dollarzeichen erfolgen soll. Sogar eine exponentielle Darstellung kann auf diese Weise erzwungen werden. Die Formatierung der Eingabe ist beschränkt auf die frei definierbare Länge des Erfassungsfeldes. Sehen wir uns eine solche INPUT-Anweisung etwas näher an:

```
INPUT (4,1) A$
```

Neu ist für den Commodore-Programmierer die Zahlenkombination in den Klammern. Die Zahl vier begrenzt die Eingabe auf maximal vier Zeichen. Durch die Zahl eins wird eine Zeitverzögerung eingestellt. Der Anwender hat dadurch etwa fünf Sekunden Zeit, um seine Eingabe durchzuführen, danach arbeitet das Programm von selbst weiter. Sind die vier möglichen Zeichen erfaßt, hängt Nevada-Basic selbständig ein <RETURN> an und arbeitet dann ganz normal weiter.

Für den maschinenorientierten Basic-Programmierer stehen wie üblich die Funktionen PEEK und POKE zur Verfügung. Damit jedoch nicht genug. Mit dem Befehl LOAD können Sie Maschinenprogramme in den Interpreter laden. Die Anfangsadresse wird dabei in eine Variable übergeben. Sobald die Maschinenroutine benötigt wird, erfolgt der Aufruf über CALL. Dem CALL-Befehl werden die Startadresse und ein zu übergebender Wert als Variablen mitgeteilt.

Eine besondere Stellung nimmt die Anweisung SYSTEM ein. Damit können Systemparameter eingestellt, ein Disketten-Reset durchgeführt oder auch alle offenen Dateien geschlossen werden.

Als Interpreter kann Nevada-Basic aufgrund des guten Preis-Leistungsverhältnisses ohne Bedenken weiterempfohlen werden. Die gute und bedienerfreundliche Programmierumgebung, wie auch der umfangreiche Sprachumfang bieten dem ernsthaften Basic-Programmierer die Grundlage für die Erstellung von sinnvollen Programmen. Das Handbuch liegt zwar nur in englischer Sprache vor, ist aber erfreulich umfangreich und sogar zu Lernzwecken geeignet. Das Fehlen von Strukturanweisung ist zwar zu bemängeln, wenn man aber bedenkt, daß Basic nicht für die strukturierte Programmierung entwickelt wurde, kann man dies ohne weiteres verschmerzen.

Die Programmiersprache Cobol wurde in den 60er Jahren im Auftrag des Pentagon in den USA entwickelt. Bis 1974 wurde die Sprache immer weiter verbessert, bis dann der sogenannte ANSI-Standard (ANSI74 und ANSI80) auf den Markt kam, der sich bis heute gehalten hat. Hauptsächlich wurde und wird Cobol auf Großrechnern in der Verwaltung zur Lösung kaufmännischer Probleme eingesetzt. Obwohl die Sprache, verglichen mit moderneren wie Pascal als veraltet gilt, findet Cobol in den klassischen Anwendungsgebieten immer noch Verbreitung. Da Cobol als Großrechnersprache entwickelt wurde, ist es nur sehr schwierig, vollständige Implementationen auf Personal Computern zu installieren.

Nevada-Cobol – der Altmeister

Wenn dies doch gelingt, lassen sich die Hersteller der Compiler ihre Errungenschaft mit hohen Preisen versilbern. Nevada-Cobol dagegen bietet zum selben Preis wie Nevada-Basic eine ansprechende Untermenge (Bild 3) des ANSI-Standards, mit dem sich, unter Verzicht einiger wichtiger Funktionen versteht sich, durchaus vernünftig arbeiten läßt. Doch zuerst einiges über die dem C 128-Besitzer vielleicht noch unbekannte Sprache.

Ein Cobol-Programm ist eingeteilt in mehrere »DIVISIONS«, die fest vorgegeben sind und zum fehlerfreien Compilieren

Nevada-Basic im Überblick			
APPEND	FNEND	ON..EXIT	ERR
BYE	DIM	ON..GOSUB	EXP
CAT	END	ON..RESTORE	FN
CLEAR	ERRCLR	OUT	FREE
CONT	ERRSET	PAUSE	INP
DEL	EXIT	POKE	INT
EDIT	FILE #n	PRINT	LEN
GET	FILL	PURGE	LOG
KILL	FNEND	READ	LOG10
LIST	FOR	REM	PEEK
LLIST	NEXT	RESTORE	POS
REN	GOSUB	REWIND	RND
RUN	GOTO	SEARCH	SGN
SAVE	IF..THEN	STOP	SIN
SCRATCH	ELSE	WAIT	SQR
SET	INPUT	ABS	STR
XEQ	LET	ASC	SYST
CLOSE	LOAD	ATN	TAB
CURSOR	LPRINT	CALL	TAN
DATA	MAT	CHR	TYP
DEF FN	ON..ERRSET	COS	VAL
RETURN	ON..EXIT	EOF	

Bild 1. Sprachumfang von Nevada-Basic

```
 10 ERASE
 20 PRINT "DATEINAME";
 30 INPUT DATEI$
 40 FILE #1;DATEI$,3,,50
 60 FOR I=1 TO 10 STEP 1
 70   ERASE
 80   PRINT "TEXT ";I;": ";
 90   INPUT TEXT$
100   PRINT #1,I;TEXT$
110 NEXT I
120 CLOSE #1
```

Bild 2. Strukturiertes Nevada-Basic-Listing

unbedingt vorhanden sein müssen. Eine Division kann man sich als Programmblock vorstellen, in dem bestimmte Aufgaben, beispielsweise die Variablendefinition, abgewickelt werden. Diese Divisions sind wiederum in »SECTIONS« unterteilt, die allerdings teilweise nur optional anzuwenden sind. In der »Identification Division« werden nur Kommentare angegeben. Hier steht beispielsweise der Programmname, der Autor und das Erstellungsdatum. Als nächstes folgt dann die »Environment Division«, die alle Informationen zur verwendeten CPU enthält. Hier werden auch die nötigen Anweisungen zur weiteren Dateihandhabung eingebaut. In der folgenden »Data Division« müssen alle im weiteren Programmverlauf verwendeten Felder definiert werden. Jedem verwendeten Feld wird eine Hierarchienummer vorangestellt. Diese ist entscheidend für die Stellung des Feldes bei der Verarbeitung. In der »Procedure Division« steht dann erst das endgültige Programm. Wie ein Cobol-Listing aussieht, sehen Sie an einem kleinen Beispiel (Bild 4). Als Source- und Object-Computer wurde eine Z80-CPU angegeben, Sie könnten aber genauso die 8080-CPU angeben, diese Zeile hat nur Dokumentationscharakter. In der nun folgenden »FILE-CONTROL« wird dem Computer mitgeteilt, daß sich die verwendete Datei auf Diskette befindet und der Zugriff sequentiell erfolgt. In der »FILE SECTION« wird die verwendete Datei näher beschrieben (FD = File Description = Dateibeschreibung). In »DAT-NAME« wird später der Name der Datei stehen und »DAT-SATZ« repräsentiert den dazugehörigen Satzaufbau. Dieser wird anschließend als Datenfeld genau definiert. In diesem Beispiel besteht er aus den beiden Variablen »FELD1« und »FELD2«. In der »Working-Storage Section« schließlich wird noch der verwendete Dateiname definiert. Die VALUE-Anweisung weist dieser Variablen sofort den gewünschten Inhalt zu. Natürlich könnte man diesen Platzhalter während des Programms neu belegen. Damit wären alle für das Programm benötigten Definitionen getätigt. In der »PROCEDURE DIVISION« wird als erstes die Datei »A:TEST.DAT« als Ausgabedatei geöffnet. Danach werden die beiden Felder des Datensatzes von der Console eingelesen. Zum Schluß wird der Satz geschrieben und die Datei geschlossen.

Das Beispielprogramm wurde mit dem Nevada-Cobol-Compiler erstellt. Ein besonderes Merkmal dieses Software-Produkts ist die überraschend hohe Geschwindigkeit. So dauert die Übersetzung des besprochenen Programmes nur 57 Sekunden. Wenn man bedenkt, daß das Commodore-CP/M nicht gerade zu den schnellsten Implementationen zählt und die Komplexität von Cobol betrachtet, ist das eine erstaunliche Leistung.

Daß Cobol eine kaufmännische Sprache ist, wissen Sie jetzt. Doch dazu ist noch einiges mehr nötig als die Verarbeitung von sequentiellen Dateien. Im kaufmännischen Bereich muß ein Programm sowohl Strings als auch Zahlen bearbeiten können. Für die String-und Zahlenbehandlung finden sich in Nevada-Cobol einige ausgereifte Befehle, die mit dem normierten Standard der Sprache identisch sind. Der eigentlich wichtigste Befehl ist MOVE. Mit diesem werden die Inhalte von Feldern ausgetauscht. Der MOVE-Befehl kann auf Strings und Zahlen angewandt werden. Nehmen wir an, Sie haben sich in der Working-Storage Section folgende Werte definiert:

```
01 STRING-1     PIC X(10).
01 STRING-2     PIC X(10).
01 ZAHL-1       PIC 999V99.
01 ZAHL-2       PIC 999V99.
```

Nun können Sie mit Hilfe des MOVE-Befehls entweder Werte zuweisen oder die Werte austauschen.

```
MOVE '64-ER' TO STRING-1.
MOVE STRING-1 to STRING-2.
```

Die Befehle von Nevada-Cobol

ACCEPT	COMP	INSTALLATION	NOT
ACCESS	COMPUTATIONAL	INTO	NUMERIC
ADD	CONFIGURATION	INVALID	OCCURS
ADVANCING	CONATAINS	IS	OF
AFTER	COPY	JUST	OMITTED
ALL	CURRENCY	JUSTIFIED	ON
ALPHABETIC	FD	KEY	OPEN
ALTER	FILE	LABEL	OR
AND	FILE-CONTROL	LEADING	ORGANIZATION
ARE	FILLER	LEFT	OUTPUT
AREA	FIRST	LESS	PAGE
ADDIGN	GIVING	LINE	PERFORM
AT	GO	LINES	PICTURE
AUTHOR	GREATER	LINKAGE	PRECEDURE
BEFORE	HIGH-VALUE	LOW-VALUE	PROCEED
BLOCK	I-O	MEMORY	PROGRAM
BY	IDENTIFICATION	MODE	PROGRAM-ID
CALL	IF	MODULES	QUOTE
CANCEL	INITIAL	MOVE	RANDOM
CHARACTARS	INPUT	MULTIPLY	READ
CLOSE	INPUT-OUTPUT	NEXT	RECORD
COMMA	INSPECT	NO	REDEFINES
RELATIVE	REPLACING	REWRITE	RIGHT
ROUNDED	RUN	SAME	SECTION
SECURITY	SELECT	SENTENCE	SEQUENTIAL
SIGN	SIZE	SPACE	STOP
SUBTRACT	SYNC	TABLE	THAN
THROUGH	TIMES	TO	UNTIL
USAGE	USING	WITH	WORDS
WRITE	ZERO		

Bild 3. Der umfangreiche Sprachschatz von Nevada-Cobol

```
MOVE 666 TO ZAHL-1.
MOVE ZAHL-1 TO ZAHL-2.
```

Sie weisen hier der Textvariablen »STRING-1« zunächst eine Konstante zu und übertragen diese dann auf die Stringvariable »STRING-2«. Genauso wird mit den Zahlenvariablen »ZAHL-1« und »ZAHL-2« verfahren. Nun wollen wir uns die Definition dieser Variablen etwas näher ansehen. Die Ziffernkombination »01« ist die Ordnungsnummer der Variablen. Darauf wird später noch genauer eingegangen. »PIC« steht für »PICTURE« und gibt den genauen Aufbau der Variablen an. So ist genau festgelegt, daß die Variablen »STRING-1« und »STRING-2« genau zehn Byte lang sind. Das »X« kennzeichnet die Variable als alphanumerisches Feld. Etwas schwieriger scheint die Definition der beiden Zahlen-Variablen zu sein. Daß dem nicht so ist, werden Sie gleich feststellen. Die neun steht hier für Zahl, und gibt dem Compiler somit unmißverständlich zu erkennen, daß es sich um numerisches Feld handelt. Der Buchstabe »V« besagt lediglich, daß an dieser Stelle das Dezimalkomma stehen soll. Die Definition könnte noch ein anderes Aussehen haben:

```
01 ZAHL-1     PIC 9(3)V9(2).
01 STRING-1   PIC XXXXXXXXXX.
```

Sie sehen also, daß man entweder mit der gewünschten Anzahl an »9« oder »X« auffüllt oder eben diese Menge als geklammerte Zahl angibt.

Beim MOVE-Befehl muß noch beachtet werden, daß die Länge des Empfangsfeldes (Feld rechts von TO) nicht kleiner sein darf als die des Feldes, von dem gesendet wird. Auch können keine alphanumerischen Daten in numerische Felder gebracht werden.

Cobol und Dateien

Bei der Dateiverwaltung fühlt sich Cobol so richtig in seinem Element. Schließlich war das einer der Hauptaspekte bei der Entwicklung der Sprache. Wie eine Datei definiert werden muß, haben Sie bereits in dem kleinen Beispielprogramm gesehen. Bei Verwendung von relativen Dateien können sich die Vorteile von Cobol erst so richtig entfalten. Soll mit einer

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003         TEST.
0004 AUTHOR.
0005         64-ER.
0006 ENVIRONMENT DIVISION.
0007 CONFIGURATION SECTION.
0008 SOURCE-COMPUTER.
0009         Z80-CPU.
0010 OBJECT-COMPUTER.
0011         Z80-CPU.
0012 INPUT-OUTPUT SECTION.
0013 FILE-CONTROL.
0014       SELECT DATEI ASSIGN TO DISK
0015       ORGANIZATION IS SEQUENTIAL
0016       ACCESS MODE IS SEQUENTIAL
0017       RECORD DELIMITER IS STANDARD.
0018*
0019 DATA DIVISION.
0020 FILE SECTION.
0021 FD DATEI
0022     LABEL RECORDS ARE STANDARD
0023     VALUE OF FILE-ID IS DAT-NAME
0024     DATA RECORDS ARE DAT-SATZ.
0025*
0026 01 DAT-SATZ.
0027     02 FELD1              PIC X(10).
0028     02 FELD2              PIC X(10).
0029*
0030 WORKING-STORAGE SECTION.
0031 01 DAT-NAME             PIC X(14)
0032                      VALUE IS "A:TEST.DAT"
0033*
0034 PROCEDURE DIVISION.
0035 BEGIN.
0036     OPEN OUTPUT DATEI.
0037     DISPLAY "FELD1: "WITH NO ADVANCING.
0038     ACCEPT FELD1.
0039     DISPLAY "FELD2: "WITH NO ADVANCING.
0040     ACCEPT FELD2.
0041*
0042     WRITE DAT-SATZ.
0043     CLOSE DATEI.
0044 STOP RUN.
0045 END PROGRAM TEST.
```

Bild 4. Bearbeiten einer sequentiellen Datei mit Nevada-Cobol

relativen Datei gearbeitet werden, müssen dazu in der File-Control noch spezielle Angaben gemacht werden.

```
FILE-CONTROL.
    SELECT TEST ASSIGN TO DISK
        ORGANIZATION IS RELATIVE
        ACCESS MODE IS RANDOM
        RELATIVE KEY IS SCHLUESSEL.
```

Hier wird wieder angegeben, daß sich die Datei auf Diskette befindet. Die Organisation ist diesmal relativ und es wird wahlfrei (relativ) zugegriffen. Der Zugriffsschlüssel ist in der Working-Storage Section unter dem Namen »SCHLUESSEL« definiert. Beim OPEN-Befehl können drei verschiedene Modi eingestellt werden: OUTPUT (nur Schreiben), INPUT (nur Lesen) und I-O (Schreiben und Lesen). Die Form der OPEN-Anweisung lautet daher wie folgt:

```
OPEN Dateiname Modus.
```

Als Dateiname wird hier jeweils der in der File-Control verwendete Name benutzt, in diesem Fall also »TEST«. Als Index zum Dateizugriff wird die jeweilige Satznummer in das Feld »SCHLUESSEL« gebracht (MOVE). Dieses Feld muß natürlich als numerische Variable definiert sein. Ein Zugriff auf die Datei sieht dann so aus:

```
MOVE 234 TO SCHLUESSEL.
READ TEST INVALID KEY GO TO FEHLER.
```

Hier soll der Satz mit der Nummer 234 gelesen werden. Sollte dies nicht möglich sein, wird automatisch zum Sprungziel »FEHLER« verzweigt. Ansonsten kann mit dem Satz gearbeitet werden, wie er in der File-Description beschrieben wurde. Das Schreiben mit dem WRITE-Befehl funktioniert im Prinzip genauso, nur daß dann mit dem Satznamen und nicht mit dem Dateinamen gearbeitet werden muß. Der Satzname ist in der File-Description festgelegt.

Tabellen und gegliederte Felder

Jeder Programmierer kennt das Problem der Tabellenverarbeitung, wenn größere Datenmengen anfallen. Die Lösung, die Cobol hierzu anbietet, ist nicht nur leistungsfähig, sondern auch komfortabel. Betrachten Sie folgendes Beispiel:

```
01 TABELLE OCCURS 600.
   02 FELD-1     PIC 9(6).
   02 FELD-2     PIC X(20).
01 TAB-FELD.
   02 TFELD-1    PIC 9(6).
   02 TFELD-2    PIC X(20).
```

Eine Tabelle wird mit der OCCURS-Klausel festgelegt. Im obigen Beispiel beläuft sich die Größe auf 600 Elemente. Jedes Element besteht aus zwei Datenfeldern (FELD-1, numerisch, neun Byte und FELD-2, alphanumerisch, 20 Byte). Diese Tabelle wird mit einem Index angesprochen. Im obigen Beispiel sind Sie auch mit der Untergliederung von Feldern konfrontiert. »TABELLE« ist ein Oberbegriff für »FELD-1« und »FELD-2«. »TAB-FELD« ist ein Oberbegriff für »TFELD-1« und »TFELD-2«. In folgendem Beispiel wird die Bedeutung der Untergliederung gezeigt:

```
MOVE 4500 TO TFELD-1.
MOVE '64-ER MAGAZIN' TO TFELD-2.
MOVE TAB-FELD TO TABELLE (2).
```

Die beiden Teile der Variablen »TAB-FELD« werden hier mit Daten versorgt. Dann wird »TAB-FELD« als Ganzes in das zweite Element der Tabelle gebracht. Durch diese Methode spart sich der Programmierer eine MOVE-Zeile und somit Speicherplatz und Rechenzeit.

Nun haben Sie einen Überblick über die Datei- und Variablenhandhabung unter Cobol. Da aber für eine kaufmännische Anwendung, wie bereits erwähnt, auch Berechnungen notwendig sind, werden noch die Rechenfunktionen der Sprache Cobol vorgestellt.

Die Sprache bietet zwei Möglichkeiten zum Programmieren von Berechnungen. Da ist zum einen der umständlichere Weg über eigene Befehle oder die Berechnung mit dem Befehl COMPUTE. Sehen wir uns zunächst die umständlichere Möglichkeit etwas näher an. Nehmen wir an, folgende Felder sind definiert:

```
01 ZAHL1      PIC 9(4).
01 ZAHL2      PIC 9(4).
01 ERGEBNIS   PIC 9(6).
```

Danach wird folgende Berechnung angestellt:

```
ADD ZAHL1 TO ZAHL2 GIVING ERGEBNIS.
MULTIPLY ZAHL1 BY ERGEBNIS GIVING ZAHL2.
DIVIDE ERGEBNIS BY ZAHL2 GIVING ZAHL1.
SUBTRACT ZAHL1 FROM ERGEBNIS GIVING ZAHL2.
```

Hier wird erst addiert, dann multipliziert, dividiert und subtrahiert. Das Ergebnis der Operation steht jeweils in dem Feld, das mit GIVING zugewiesen wird. Wie bereits erwähnt, kann dies auch mit der COMPUTE-Anweisung durchgeführt werden. Damit sieht die Berechnung dann so aus:

```
COMPUTE ERGEBNIS = ZAHL1 + ZAHL2.
COMPUTE ZAHL2 = ERGEBNIS * ZAHL1.
COMPUTE ZAHL1 = ERGEBNIS / ZAHL2.
COMPUTE ZAHL2 = ERGEBNIS - ZAHL1.
```

Diese Schreibweise ist für den Basic-Programmierer wesentlich verständlicher als die vorhergehende.

Man sieht also ganz deutlich, daß sich Cobol von anderen Sprachen abhebt. Zum einen sind genaueste Definitionen der verwendeten Variablen notwendig, zum anderen sind die Anweisungen und Befehle so formuliert, daß sich ein Cobol-Programm beinahe wie ein Roman liest.

Im Lieferumfang von Nevada-Cobol ist noch zusätzlich ein Renumber-Programm enthalten. Die Programme müssen mit vierstelligen Zeilennummern geschrieben werden. Da die Texteditoren, die für CP/M erhältlich sind, über keine Zeilennummern-Funktionen verfügen, müssen derartige Aktionen über externe Programme durchgeführt werden. Der Compiler selbst bietet noch einen »Debugging-Mode«, der eine komfortable Fehlersuche ermöglicht.

Trotz der hohen Ansprüche, die der ANSI-74-Standard an die Sprache Cobol stellt, konnte bei Nevada-Cobol eine erstaunlich große Untermenge dieses Standards implementiert werden.

Das größte Manko von Cobol besteht in den fehlenden Strukturanweisungen, wie sie heute in allen modernen Sprachen vorhanden sind. Die einzigen Schleifenanweisungen sind IF..THEN und FOR..NEXT. Auch die Komplexität der Sprache wird viele Programmierer abschrecken, Cobol anzuwenden. Wenn man allerdings längere Zeit damit gearbeitet hat, möchte man die komfortablen Elemente zur Dateiverwaltung nicht mehr missen. Hier bieten auch moderne Sprachen, wie zum Beispiel Pascal, keine revolutionären Verbesserungen.

Für weitere 29 Mark erhalten Sie ein Buch mit mehreren Cobol-Programmen zum Abtippen, unter anderem ein Haushaltsbuch. Das Handbuch zu Nevada-Cobol ist erfreulich umfangreich und beinhaltet unter anderem eine Einführung in die Programmiersprache Cobol. Es ist in deutsch und in englisch erhältlich.

Abschließend kann bemerkt werden, daß sich der Cobol-Compiler optimal für den Einsteiger in diese Sprache, wie auch für den ernsthaften CP/M-Programmierer eignet. Unterstützt wird diese Aussage durch das sehr gute Preis-Leistungsverhältnis, das momentan hauptsächlich auf den schwachen Dollarkurs zurückzuführen ist.

Nevada-Pascal – Leistung ist alles

Mehr und mehr hat sich in letzter Zeit Pascal als die Programmiersprache für Mikro-Computer durchgesetzt. Beinahe ein Standard sind Compiler wie zum Beispiel Turbo-Pascal geworden. Daneben gibt es auch noch eine ganze Reihe anderer Produkte, deren Leistung nicht zu verachten ist. Ein wie bei der Nevada-Serie bereits gewohnt sehr kostengünstiges Produkt wird mit Nevada-Pascal angeboten.

Mit der Programmiersprache Pascal werden wir uns hier nicht näher befassen. Dazu finden Sie in dieser Ausgabe einen ausführlichen Kurs und zwei weitere Compiler-Beschreibungen.

Nevada-Pascal weist einige Besonderheiten auf, die sich sehr vorteilhaft auf das Produkt auswirken. Zum einen wäre da eine sehr schnelle und komfortable Tabellensuch-Funktion, zum anderen die Möglichkeit, ohne große Programmiertricks index-sequentielle Dateien zu erstellen. Bei dieser Dateiart wird nicht über die Satznummer auf die Datei zugegriffen, sondern über einen Suchbegriff (String). Darauf wollen wir im folgenden näher eingehen. Um index-sequentielle Dateien zu erstellen, ist es normalerweise notwendig, die

Nevada-Pascal Funktionen			
ABS	ADDR	ARCTAN	CHR
CONCAT	COPY	COS	EXP
FREE	HEX$	LENGTH	LN
ODD	ORD	PORTIN	POS
PRED	REAL$	ROUND	SEARCH
SIN	SQR	SQRT	SUCC
TRUNC	UPCASE	CALL	DELETE
DISPOSE	FILLCHAR	INSERT	MAP
NEW	PORTOUT	SYSTEM	RESET
REWRITE	CLOSE	READLN	WRITELN
EOF	EOLN	ERASE	RENAME
GET	PICTURE	PUT	

Bild 5. Die Funktionen von Nevada-Pascal

Fortran	Basic 7.0	Bedeutung
.EQ.	=	Gleichheit
.NE.	<>	Ungleichheit
.GT.	>	Größer
.LT.	<	Kleiner
.GE.	>=	Größer oder gleich
.LE.	<=	Kleiner oder gleich
.NOT.	NOT	Logisch nicht
.AND.	AND	Logisch und
.OR.	OR	Logisch oder
.XOR.	XOR	Logisch exklusiv oder

Bild 6. Fortran-Operatoren im Vergleich mit Basic 7.0

Verwaltung der Index-Datei selbst zu programmieren. Dadurch wird das Programm aufgebläht und sehr schnell unübersichtlich. Zwar kennt Nevada-Pascal keine implementierten Funktionen und Prozeduren zum Aufbau einer solchen Dateiverwaltung, dafür werden aber auf der Compiler-Diskette einige Programmdateien mitgeliefert, die über eine »EXTERN«-Deklaration in das Pascal-Programm eingebunden werden. Einige Variablendefinitionen sind allerdings notwendig, um die benötigten Parameter übergeben zu können. Am Anfang ist das Arbeiten mit diesen Prozeduren sicherlich etwas ungewohnt und umständlich. Wer aber selbst schon eine index-sequentielle Dateiverwaltung programmiert hat, wird die Nevada-Lösung bald zu schätzen wissen. Die Auswahl der Kommandos, die für die index-sequentielle Bearbeitung zur Verfügung stehen, läßt keine Wünsche offen. Vom Schreiben eines Records bis zum Anfügen von neuen Sätzen ist alles implementiert. Dabei wird die erstellte Indexdatei ständig aktualisiert.

Die bereits angesprochene schnelle Routine zum Durchsuchen von Tabellen steht als Extern-Funktion zur Verfügung. Nachdem die nötigen Parameter im Definitionsteil deklariert sind, können problemlos große Arrays nach bestimmten Kriterien (Keys) durchsucht werden.

Für die Fehlersuche in fertigen Programmen (Debugging) kann mit Hilfe der Funktion »TRACE« jede Programmzeile während des Ablaufs am Bildschirm angezeigt werden. Der Compiler erlaubt mehrere Trace-Modi. So kann man wählen zwischen Zeilen-, Prozedur- und Funktions-Trace. Zur Gestaltung des Ausgabe-Listings, sowie der Festlegung der Ausgabeeinheit sind weitere Compiler-Optionen vorhanden.

Da bei der Compilierung kein Run-Time-Modul erzeugt wird, erfolgt der Aufruf eines jeden Nevada-Pascal-Programms über ein eigenes Startprogramm. Auch hier können noch zusätzliche Optionen angegeben werden. So kann man nachträglich den Trace-Modus einschalten, oder ein »Activity-Analyser« (Analysieren der Programmaktivitäten).

Nevada-Fortran: Sprachumfang			
ACCEPT	ASSIGN	BACKSPACE	BLOCK DATA
CALL	COMMON	CONTINUE	COPY
CTRL DISABLE	CTRL ENABLE	DATA	DIMENSION
DO	DOUBLE	DUMP	END
ENDFILE	ERRCLR	ERRSET	FORMAT
FUNCTION	GO TO	IF..THEN..ELSE	IMPLICIT
INTEGER	LOGICAL	PAUSE	READ
REAL	RETURN	REWIND	STOP
SUBROUTINE	TRACE OFF	TRACE ON	TYPE
WRITE	SIN	COS	TAN
ATAN	ATAN2	ALOG	ALOG10
MOD	AMOD	SQRT	FLOAT
IFIX	ABS	IABS	RAND
EXP	COMP	CALL	AMAXO
AMAX1	MAXO	MAX1	AMINO
AMIN1	MINO	MIN1	BIT
SIGN	ISIGN	DIM	IDIM
CHAR	CALL	INP	CBTOF
PEEK			
Systemroutinen, die mit CALL aufgerufen werden:			
BIT	CBTOF	CHAIN	CIN
CLOSE	CTEST	DELAY	DELETE
EXIT	MOVE	OPEN	LOAD
LOPEN	POKE	OUT	PUT
RENAME	RESET	SEEK	SETIO

Bild 7. Nevada-Fortran-Funktionen im Überblick

Nevada-Pilot: Befehle und Anweisungen					
T:	A:	M:	J:	U:	E:
C:	R:	APPEND:	BYE	CA:	CALL:
CE:	CH:	CL:	CLOSED:	CRAETEF:	DIR:
EDIT	EOF:	FOOT:	GET	INFO	INMAX:
KILLF	LIST	LOAD	NEW	OPENF:	PAUSE:
POKE:	PR:	READ:	REWIND:	RL:	RUN
SAVE	SET:	VIEW:	VNEW:	VSET:	VSTART:
WRITE:	XEQ	XI:	XS:		

Bild 8. Die ungewöhnlichen Anweisungen von Nevada-Pilot

Diese Analyse-Funktion erlaubt einen direkten Eingriff in den Programmablauf. Man kann damit alle Systemzeiger auf Null stellen oder ein Aktivitätshistogramm ausgeben lassen.

Alles in allem zeigt sich der Nevada-Pascal-Compiler als sehr flexibles Software-Produkt mit herausragenden Leistungsmerkmalen, die man in dieser Preisklasse nur selten findet. Die Möglichkeiten, die sich aus den vielfältigen Funktionen (Bild 5) des Compilers ergeben, lassen dem Programmierer große Freiräume bei der Erstellung von umfangreichen Dateiverwaltungen.

Nevada-Fortran für Mathematiker

Die Sprache Fortran findet hauptsächlich im wissenschaftlichen Bereich Anwendung. Wenn umfangreiche und komplexe Berechnungen notwendig sind, bildet Fortran durch seine mathematische »Begabung« ein optimales Werkzeug für schnelle Computerlösungen. Wie Cobol, so ist auch Fortran eine bereits etwas ergraute Sprache, die fast nur noch auf Großrechenanlagen eingesetzt wird. Der größte Nachteil von Fortran ist zweifelsfrei das benötigte Quellcode-Format. Hier muß man sich strikt an die vorgegebenen Regeln halten, jede Abweichung vom Standard nimmt der Compiler sofort übel. Die ersten sechs Spalten einer Programmzeile dürfen nie mit Fortran-Anweisungen beschrieben werden, da diese Stellen allein für Label vorgesehen sind. Fortran-Label werden immer durch Zahlen ausgedrückt, können also keine konkreten Namen haben. Die Ein- und Ausgabeformate sind nur sehr umständlich zu handhaben, dafür aber relativ universell. So kann die Eingabe wie auch die Ausgabe von Daten ohne weiteres auf ein anderes Gerät umgelenkt werden. Die klassischen Geräte waren hier der Lochkartenleser und -stanzer. Die strukturierte Programmierung ist für den Fortran-Programmierer nur schwer zu realisieren.

Die im Programmverlauf verwendeten Variablen müssen am Anfang definiert werden. Dabei wird auch der Variablentyp mit angegeben (zum Beispiel: Integer). Um das Vorurteil der Unstrukturiertheit nicht ganz auf Fortran sitzen zu lassen, muß zu dessen Ehrenrettung gesagt werden, daß einfache Schleifenstrukturen, wie DO-WHILE...END-DOWHILE und IF..THEN...ELSE sehr wohl vorhanden sind. Der erfahrene Pascal-Programmierer wird allerdings die leistungsfähigen und bequemen Prozeduren und Funktionen vermissen. Allerdings können Unterprogramme definiert werden, denen man verschiedene Parameter übergeben kann.

Im weiteren werden wir etwas näher auf die mathematischen Fähigkeiten von Fortran eingehen. Für die Verarbeitung existieren zwei Datentypen: Integer (ganze Zahlen) und Real (Gleitkommazahlen). Zum Rechnen stehen natürlich die vier Grundrechenarten zur Verfügung, zusätzlich noch Potenzieren. Die verschiedenen, auch aus Basic bekannten Vergleichsoperatoren werden in Fortran nicht durch die bekannten Zeichen dargestellt, sondern durch entsprechende Zeichenoperatoren. So lautet beispielsweise der Operator für Ungleich: ».NE.« (not equal to). Die Operatoren müssen in Punkte eingeschlossen werden, um vom Compiler als Operatoren erkannt zu werden. Alle Standardoperatoren sind vorhanden (Bild 6). Für die Bearbeitung von mathematischen Problemen bietet Nevada-Fortran eine umfangreiche Standard-Bibliothek (Bild 7), die im wesentlichen dem ANSI-Standard entspricht. Zur Berechnung von Bogengraden stehen vier Funktionen zur Verfügung. Der Logarithmus kann als natürlicher oder 10er-Logarithmus berechnet werden. Mit einer weiteren Anweisung können die Nachkommastellen eines Divisionsergebnisses bearbeitet werden.

Auch für den Mathematiker tut sich das Problem der Datenspeicherung auf. Dazu bietet Nevada-Fortran eine ganze Reihe von Anweisungen an, die mit dem Befehl CALL aufgerufen werden. Die Definition von Feldern ist mit Nevada-Fortran ebenfalls möglich.

Wer seine mathematischen Probleme mit einer klassischen Programmiersprache lösen möchte, ist mit Nevada-Fortran bestens bedient. Nicht nur seine Leistungsfähigkeit macht dieses Produkt interessant, sondern auch der gewohnt niedrige Preis, der es dem Heimanwender erlaubt, auf die Sprache seiner Wahl zurückzugreifen, ohne einen größeren Kredit aufnehmen zu müssen. Wer tiefer in die Tasche greifen will, kann natürlich auf ein großes Compiler-Angebot zurückgreifen, muß sich aber auf Preise oberhalb der 1000-Mark-Grenze gefaßt machen.

Nevada-Pilot für Individualisten

Entwickelt wurde Pilot zum Entwerfen von Lernprogrammen, die nach dem Frage-und-Antwort-Prinzip arbeiten. Zweifellos eignet sich die Sprache durch die kurz gehaltene Schreibweise der einzelnen Befehle hervorragend für diese Programmart. Der Programmierer muß sich nicht mit Syntax-Regeln abmühen, sondern kann sich auf den Kern des Problems konzentrieren. Für das bei Lernprogrammen auftretende Problem der String-Verarbeitung bietet Pilot eine Reihe von leistungsfähigen und nicht alltäglichen Befehlen (Bild 8) an.

Pilot-Anweisungen sind nach folgendem Prinzip aufgebaut:

Befehl:Operandenliste

Einer der wichtigsten Pilot-Befehle ist die Type-Anweisung.

Diese existiert in zwei Formen. Die normale Type-Anweisung sieht folgendermaßen aus:

```
T:PILOT IST SPITZE
```

Dadurch erscheint ganz einfach der Text nach dem Doppelpunkt auf dem Bildschirm. Um Eingaben direkt an den Text anzuschließen wird der Befehl TH verwendet. Mit der Anweisung TP läßt sich der nachfolgende Text auf den Drucker umlenken. Mit einem ebenso einfachen Befehl werden Daten vom Bildschirm eingelesen. Die dem aus Basic bekannte INPUT-äquivalente Anweisung lautet in Pilot »A:« (Accept = akzeptiere). Jetzt sind praktisch die beiden wichtigsten Elemente von Pilot erklärt, um mit dem Computer in Verbindung zu treten. Die vom Anwender eingegebenen Daten muß man auf ihre Richtigkeit überprüfen können. Dazu bietet Nevada-Pilot wieder einige ungewohnte Anweisungen an, vergleichbar mit IF..THEN aus Basic. So können über die Anweisung »M:« verschiedene Kriterien angegeben werden. Wurde keines der definierten Kriterien eingegeben, so wird in Abhängigkeit des gewählten Sprungbefehls zum entsprechenden Label verzweigt. Diese Art der Abfrage beschränkt sich allerdings auf einige fest vorgegebene Sprungziele.

Die integrierte Sprunganweisung »J:« (Jump) ermöglicht das Ansteuern von freidefinierbaren Sprungzielen. Ein Sprungziel (Label) wird immer mit einem vorangestellten Stern (*) gekennzeichnet, auch beim Aufruf mit »J:«. Dieser Befehl ermöglicht auch den Sprung unter einer bestimmten Bedingung. Doch nun zurück zum bereits erwähnten Match-Befehl (M:). Dazu sehen wir uns ein kleines Beispiel an (Bild 9). Der Befehl in der ersten Zeile ist für das Löschen des Bildschirms verantwortlich. In Zeile zwei wird ein Label mit Namen »START« definiert. Danach wird mit dem Type-Befehl eine Frage auf dem Bildschirm ausgegeben. »A:« wartet auf eine Eingabe des Bedieners. Die M:-Anweisung legt fest, daß als gültige Eingabe nur »PARIS« zugelassen ist. Wurde PARIS eingegeben, ist die Bedingung »TY:« (Type if yes) erfüllt und es erscheint »RICHTIG« am Bildschirm. Die nächste Bedingung »JY:« (Jump if yes) ist somit ebenfalls erfüllt und es folgt ein Sprung zum Label »ENDE«. Wurde eine falsche Antwort eingegeben, werden die beiden Bedingungen als »nicht erfüllt« erkannt und übergangen. Ein kleiner Tip soll auf dem Weg zur richtigen Antwort weiterhelfen. Ist die nächste Eingabe wieder falsch, erscheint die korrekte Hauptstadt, ansonsten wird nach Ausgabe der entsprechenden Meldung zum Ende gesprungen. Die Anweisung »E:« zeigt dem Pilot-Interpreter das Programmende.

Neben diesen Standard-Routinen bietet Nevada-Pilot noch zusätzliche Funktionen für Berechnungen und die Dateiverwaltung. So können die für Lernprogramme benötigten Daten in Dateien abgelegt und bei Bedarf eingelesen werden.

C – Die Sprache der Zukunft

Derzeit wird die EDV-Welt von der noch relativ jungen Sprache C stark beeinflußt. Ursprünglich für das Betriebssystem UNIX entwickelt, steht C nun schon seit längerem für den C 128 zur Verfügung, sei es im C 128-Modus oder im CP/M-Modus. Der hier vorgestellte C/80-Compiler von Software-Toolworks läuft unter CP/M und bietet überraschend viele Standardfunktionen (Bild 10) der C-Norm, die von Kernigan/Ritchie, den »C-Erfindern«, festgelegt wurden.

Der Compiler erzeugt in einem Durchgang ein Assemblerfile und ein ablauffähiges COM-File. Zahlreiche Parameter, die beim Compileraufruf mit angegeben werden, ermöglichen es dem Programmierer die Grundeinstellung des Compilers zu verändern und somit die Geschwindigkeit des Codes, was allerdings auf Kosten des Speicherplatzes geht.

Die vom C/80 angebotenen Variablentypen brauchen den Vergleich mit anderen C-Compilern nicht zu scheuen. So stehen außer den Datentypen »LONG« und »FLOAT« alle Möglichkeiten zur Auswahl. Die fehlenden Typen sind aber dann im Erweiterungspaket »C/80 Mathpak« enthalten, das zu einem Aufpreis von 99 Mark erhältlich ist. Zeiger sind ebenfalls vorhanden und arbeiten mit allen Datentypen zusammen. Die Speicherungsklassen, die C/80 zur Verfügung stellt, entsprechen ebenfalls dem Standard. Sogar die Speicherklasse »auto« ist vertreten, die zu den Standardeinstellungen lokaler Variablen innerhalb einer Funktion gehört. Daneben gibt es noch die »static«-Klasse, die lokale »auto«-Variablen auch nach dem Verlassen einer Funktion am Leben erhält. Eine weitere Speicherklasse wird mit »register« angeboten. Normalerweise ist es dem Compiler überlassen, wo er seine Variablen ablegt. Wenn Sie die Variable mit »register« deklarieren, verarbeitet der Compiler diese Variable direkt in einem der Prozessor-Register. Somit ist der schnellstmögliche Zugriff auf die Variable gewährleistet. Zu guter Letzt können Sie noch die Speicherklasse »extern« anwenden. Damit können Variablen, die in einer als Modul verwendeten Funktion vorkommen, als global definiert werden, so daß aus allen anderen Programmebenen darauf zugegriffen werden kann.

Auffällig an C/80 ist vor allem die Tatsache, daß Standard-Dateien fehlen, die jedem C-Programmierer nach einiger Zeit vertraut sind. Gemeint ist hier beispielsweise »STDIO.H«, die ständig verwendete Ein-/Ausgabe-Parameter enthält. Die bei C/80 verwendeten »INCLUDE«-Dateien enthalten größtenteils die Module für die Steuerung der Bildschirm-, Disketten- und Drucker-Ein-/Ausgabe. So gibt es für die formatierte Bildschirmausgabe mit »PRINTF« eine eigene Datei, die jedesmal mit der »INCLUDE«-Anweisung eingebunden werden muß, sobald sie, und das ist meistens der Fall, im Programm verwendet wird. Genauso wird mit einigen Funktionen der Dateiverwaltung verfahren, die in einer Datei namens »SEEK.C« abgelegt sind. Weiterhin steht »SCANF.C« für die formatierte Ausgabe und »STDLIB.C« für Systemzugriffe und String-Behandlung zur Verfügung.

Wem C zur Systemmanipulation immer noch nicht ausreicht, der kann über die Anweisung »#ASM« Assemblercode direkt in das Programm einfügen. Die Assemblerroutine

```
ch:
*start t:Wie heißt die Hauptstadt von Frankreich?
a:
m: paris
ty:Richtig
jy:*ende
t:Vielleicht hilft Ihnen Pigalle weiter
a:
m:paris
ty:Jetzt stimmt's
jy:ende
t: Paris wäre die richtige Antwort gewesen
*ende
:e
```

Bild 9. Pilot-Listing mit einfachem Entscheidungsbaum

C/80-Funktionsübersicht			
GETCHAR()	PUTCHAR()	FOPEN()	GET()
PUT()	FCLOSE()	READ()	WRITE()
EXIT()	SBRK()	PRINTF()	SCANF()
ABS()	ATOI()	ITOA()	INDEX()
ISALPHA()	ISDIGIT()	ISLOWER()	ISUPPER()
ISSPACE()	STRCAT()	STRCMP()	STRCPY()
STRLEN()	TOLOWER()	SEEK()	FTELL()
FTELLR()	EXEC()	COMMAND()	

Bild 10. Standardfunktionen des C/80-Compilers

wird mit »#ENDASM« abgeschlossen. Beim Compilieren wird der Assemblercode übergangen und erst bei der Assemblierung weiterverarbeitet.

Vor allem für den Systemprogrammierer ist die »INCLUDE«-Datei »CLIBIO.C« interessant, in deren Definitionsteil alle Systemaufrufe vorhanden sind. Allerdings nur die Aufrufe der CP/M-Version 2.2, die nicht alle Funktionen des CP/M 3.0 enthält. Diese lassen sich, wenn der Programmierer über die nötigen Kenntnisse verfügt, ohne weiteres anfügen.

Abschließend kann gesagt werden, daß sich der C/80-Compiler nicht nur für die Systementwicklung eignet, wie es bei Small-C der Fall ist. Der enorme Befehlsumfang sowie die Vielfalt der Variablen- und Speicherklassentypen erlaubt es dem versierten C-Programmierer durchaus, ernsthafte Anwendungen zu entwickeln. Da der Compiler ein Programm in einem Durchgang compiliert, werden aktzeptable Zeiten erreicht. Zum Schluß sei noch auf die Erweiterungspakete hingewiesen. Wie bereits erwähnt, ist zusätzlich das C/80-Mathpak erhältlich, eine Erweiterung für mathematische Anwendungen. Der C/80-Compiler, der für 189 Mark erhältlich ist, wird auch zusammen mit dem Mathpak angeboten, die Kosten belaufen sich dann auf 279 Mark.

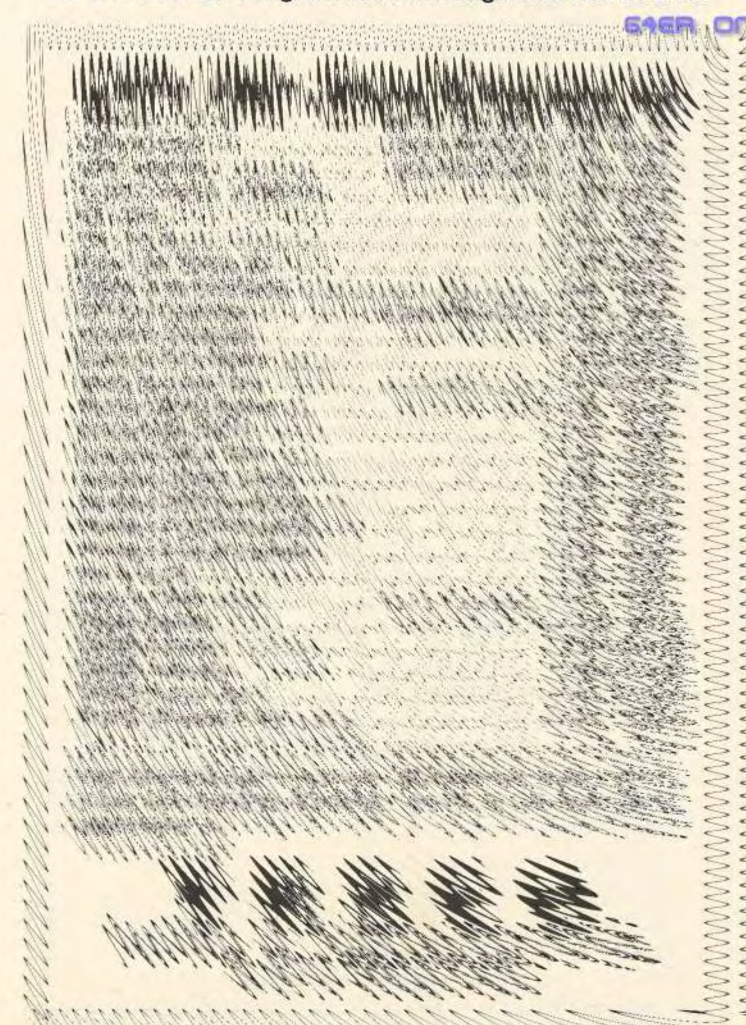
Nevada-Edit – komfortable Programmierumgebung

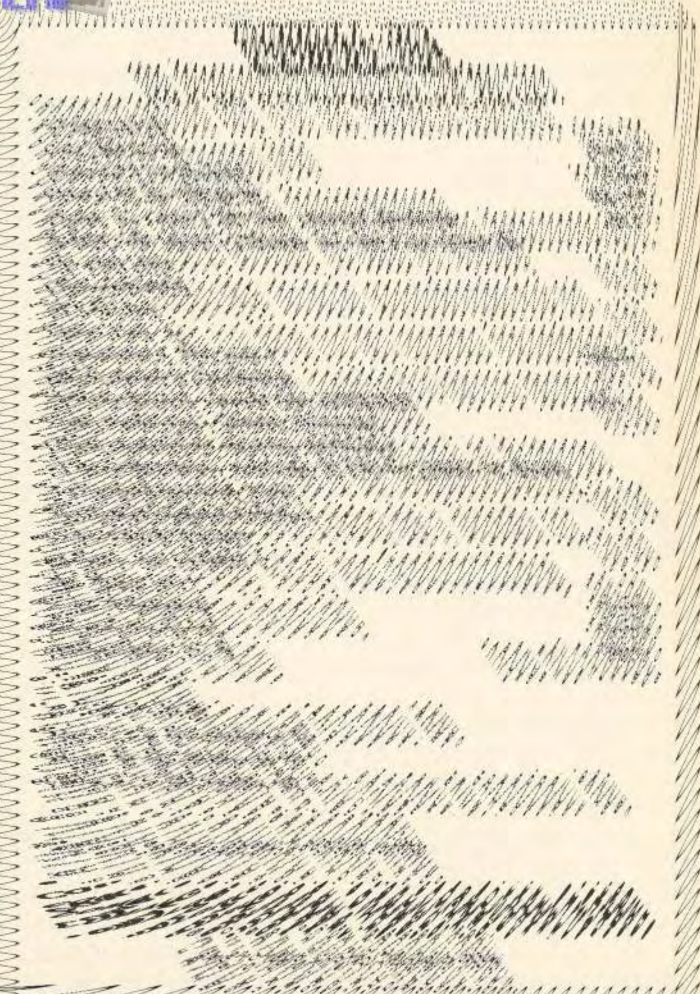
Um unter CP/M sinnvoll programmieren zu können, benötigt man einen leistungsstarken Bildschirm-Editor, der eine beliebige Cursor-Steuerung erlaubt (Full-Screen-Editor). Vom CP/M-ED wird man in dieser Beziehung bitter enttäuscht. Wer sich aber zur Programmentwicklung keine Textverarbeitung wie Wordstar anschaffen will, kann auch auf Nevada-Edit zurückgreifen. Der Editor kostet inklusive Handbuch 89 Mark. Er bietet Full-Screen-Behandlung des editierten Textes. Bei der Konfiguration des Editors an den C 128 muß wieder nur der Terminal-Typ »Lear-Seigler ASM31« angewählt werden. Danach erfolgt die Auswahl der gewünschten Extension (Dateikennung). Die editierten Programme werden ohne die Dateikennung aufgerufen. Diese wird automatisch angefügt. Allerdings muß die Extension drei Zeichen lang sein. Für den C-Compiler wäre diese Funktion also ungeeignet. Hier muß die Dateikennung beim Starten des Editors mit angegeben werden.

Der Editor selbst bietet eine reichhaltige Auswahl an Funktionen, die teilweise Wordstar-orientiert sind. Nevada-Edit erweist sich als geeignetes Werkzeug zur Programmierung mit den verschiedenen Nevada Compiler-Sprachen. Der Editor kann aufgrund seiner Funktionsvielfalt uneingeschränkt zur Erstellung von Quelltexten oder auch als Mini-Textverarbeitung verwendet werden und zeigt sich nach längerer Anwendung als große Hilfe bei der Programmierung.

Viel Sprache für wenig Geld. So könnte man das Angebot der Nevada-Serie zusammenfassen. Beinahe jede gängige Sprache ist damit im CP/M-Modus des C 128 verfügbar. Doch auch für die Liebhaber moderner Sprachen bieten beispielsweise Nevada-Pascal oder C/80 eine echte Alternative zu den sonst oft überteuerten Compiler-Sprachen, die unter CP/M erhältlich sind. Man kann zwar nicht erwarten, daß ein billiges Produkt die Leistung der großen und teuren Compiler erreicht, ist aber angenehm überrascht, wenn beim ersten Arbeiten bereits die Funktionsvielfalt der verschiedenen Compiler auffällt. (rf)

Info: Tesco GmbH, Rüdenhausenerstraße, 8714 Wiesentheid
Comfood GmbH, Flaßkamp 24, 4400 Münster





Z 80 – Der CP/M-Steuermann

Lernen Sie die zweite Seele des C128, den Z 80-Prozessor, näher kennen. Damit können Sie das leistungsfähige Betriebssystem CP/M besser ausnutzen.

Wer sich heute ernsthafter mit CP/M beschäftigen will, kommt nicht darum herum, den Z 80 Mikroprozessor etwas näher zu betrachten.

Das Betriebssystem CP/M (»C«ontrol-»P«rogramm for »M«icrocomputers), das von der Firma Digital Research entwickelt wurde, ist zwar ursprünglich für den 8080-Mikroprozessor der Firma Intel geschrieben worden, doch heutzutage wird in CP/M-Computern fast ausschließlich der Z 80-Mikroprozessor (CPU) verwendet. So wird auch im Commodore 128 PC eine Z 80 A CPU (A = 4-MHz-Version) für den CP/M-Modus eingesetzt.

Kompatibilität Z 80 – 8080

Die Kompatibilität der 80er-Prozessoren von Intel und Zilog stiftet bei CP/M-Benutzern immer wieder Verwirrung. Deshalb wird an dieser Stelle hierauf eingegangen.

Die 8080-CPU der Firma Intel kann als der Stammvater der sogenannten 80er-Familie bezeichnet werden. Diese CPU ist seit 1974 lieferbar. Die Z 80-CPU wurde von der Firma Zilog entwickelt und wird seit 1977 vertrieben. Dieser Mikroprozessor ist als konsequente Weiterentwicklung des 8080-Mikroprozessors zu sehen. Der Z 80 ist aufwärtskompatibel zum 8080-Mikroprozessor, das heißt er »versteht« alle Befehle des 8080 und besitzt darüber hinaus einen stark erweiterten Befehlssatz. Dieser erweiterte Befehlssatz darf aber für CP/M-Anwenderprogramme nicht benutzt werden, da Programme dann nicht mehr auf allen CP/M-Rechnern laufen würden (keine Kompatibilität mehr). Die Nutzung des erweiterten Befehlssatzes ist also auf Einzelanwendungen beschränkt. Einzelne Softwarehäuser weichen jedoch heute von dieser Regelung ab und bieten spezielle CP/M-Programme für CP/M- Rechner mit Z 80-CPU an. Diese Softwarehäuser müssen sich jedoch den Vorwurf der »Nichtkompatibilität« gefallen lassen.

Wer nun glaubt, er könne einen defekten Z 80 durch einen 8080 austauschen, der irrt gewaltig. Die Kompatibilität dieser beiden Prozessoren beschränkt sich auf den Befehlssatz des 8080, das heißt der Z 80 versteht die Opcodes des 8080. Hardwaremäßig besteht keine Kompatibilität. Der 8080 (und die 8080-Nachfolger der Firma Intel) benutzt einen gemultiplexten Datenbus, das heißt auf den Adreßleitungen A0-A7 werden abwechselnd Adressen und Daten »transportiert«, während beim Z 80 alle Anschlüsse einzeln herausgeführt werden.

Die Verwirrung bei ernsthaften CP/M-Einsteigern ist häufig vollends, wenn zur Erstellung von Maschinenprogrammen Assembler benutzt werden sollen.

Der Besitzer eines C 128 weiß, daß sein Computer mit einer Z 80-CPU ausgerüstet ist und das Betriebssystem CP/M Plus mitgeliefert wurde. Um den Z 80 näher kennenzulernen und um in Maschinensprache zu arbeiten, besorgt er sich ein Buch über den Z 80 und die zu CP/M Plus gehörenden Utilities SID, MAC und HEXCOM. SID ist ein Maschinensprachemonitor mit kleinem Assembler und Disassembler. MAC ist ein Makroassembler, der ein Maschinenprogramm in Assemblersprache (mnemonische Form) in den Maschinencode übersetzt. HEXCOM erzeugt aus den von MAC übersetzten Dateien Programme, die direkt vom Betriebssystem aus aufgerufen werden können (.com-Dateien). Wenn man sich jetzt mit SID einen Teil des Z 80-ROMs in seinem C 128 disassemblieren läßt, stellt man fest, daß die aufgelisteten Maschinensprachebefehle in mnemonischer Form nicht mit denen eines Z 80-Buches übereinstimmen. Das liegt daran, daß die Opcodes des Z 80 und des 8080 zwar übereinstimmen (mit den weiter oben beschriebenen Einschränkungen), jedoch für beide Prozessoren andere Mnemonics verwendet werden. Die zum CP/M-System mitgelieferten Disassembler erzeugen jedoch Mnemonics im Intel-Format (8080-Mnemonics).

Wer also den Z 80 kennenlernen möchte, muß entweder mit Opcodes arbeiten (zum Beispiel mit dem SID-Monitor) oder muß sich einen Z 80-Assembler besorgen. Die weiter unten beschriebenen Z 80-Assemblerbeispiele wurden mit dem Z 80-Makroassembler erstellt und getestet, der mit dem Small-C-Entwicklungssystem vom Markt&Technik Verlag mitgeliefert wird.

Der CP/M-Macroassembler MAC kann zwar auch Z 80-Mnemonics übersetzen, jedoch muß eine Mnemonic-Form benutzt werden, die sehr stark von der üblichen Zilog-Version abweicht, so daß dieser Weg nur als Notlösung akzeptiert werden kann. Welche Form die zu verwendenden Mnemonics haben müssen, können Sie erfahren, wenn Sie die entsprechende Diskette einlegen (CP/M Utilities-Disk 2) und dann »TYPE Z80.LIB« (<RETURN>) eingeben.

Der Z 80-Macroassembler des Small-C-Entwicklungssystems und die meisten anderen Z 80-Assembler benutzen die übliche Zilog-Version für Z 80-Mnemonics.

Diese Vorbemerkungen sind notwendig, damit die ersten Versuche, praktisch mit dem Z 80 unter CP/M zu arbeiten, nicht in Verwirrung und Frustration enden.

Aufbau des Z 80

Der Z 80-Mikroprozessor besitzt einen 8 Bit breiten Datenbus (deshalb 8-Bit-Prozessor), einen 16 Bit breiten Adreßbus und einen Steuerbus. Der Systemtakt muß extern von einem separaten Taktgenerator erzeugt werden. Der prinzipielle interne Aufbau des Z 80-Prozessors sieht wie folgt aus:

Im Inneren des Prozessors befinden sich:

- eine Arithmetik-Logik-Einheit (ALU)
- ein Steuerwerk, das die internen Abläufe steuert sowie den Steuerbus verwaltet
- vier reine Adreßregister (1. der Befehlszähler PC, 2. der Stapelzeiger SP, 3. das Indexregister IX und 4. das Indexregister IY).
- ein Refresh-Register (R-Register), das dazu dient, dynamische Speicher aufzufrischen.
- zwei Akkumulatoren (A und A'), von denen jedoch nur einer zur Zeit benutzt werden kann. Auf den zweiten Akku kann jedoch umgeschaltet werden.
- zwei Registersätze mit »normalen« Registern (B-,C-,D-,E-, H- und L-Register beziehungsweise B', C',D',E',H' und L' Register). Diese jeweils 8-Bit breiten Register können zu jeweils drei 16-Bit breiten Registern zusammengezogen werden (BC-,DE- und HL-Register).

  Auch hier ist eine Registersatzumschaltung möglich (wie bei den Akumulatoren).
- interne Daten-, Adreß- und Steuerbusse. Diese internen Busse sind durch Puffer von den äußeren Bussen getrennt.

Es existieren noch weitere interne Register, die jedoch für den Programmierer nicht direkt erreichbar sind.

Vergleicht man den Z 80 mit dem 6502, so fällt die größere Anzahl von Registern beim Z 80 auf. Der Z 80 benötigt auch mehr Register, denn er gehört zu den sogenannten registerorientierten Prozessoren (wie die gesamte 80er-Familie), der 6502 zu den speicherorientierten Prozessoren (65er- und 68er-Familie).

Viele Arbeiten, die die speicherorientierten Prozessoren im Arbeitsspeicher (RAM) durchführen, müssen bei den registerorientierten Prozessoren innerhalb des Prozessors in einem ihrer vielen Register durchgeführt werden. Die 6502 CPU behandelt Ein-/Ausgabebausteine (I/O) wie normale Speicherzellen. Das bedeutet, daß die meisten CPU-Befehle auch auf I/O-Bausteine anwendbar sind.

Die Z 80 CPU benutzt für Ein-/Ausgabe über I/O-Bausteine besondere Befehle (IN und OUT). Die I/O-Bausteine befinden sich nicht im »normalen« Adreßbereich, sondern werden über eine besondere Input/Output-Request-Leitung (IORQ) aktiviert und über die untere Hälfte des Adreßbusses (A0-A7) adressiert. Nach einem Hardware-Reset (zum Beispiel nach dem Einschalten) beginnt die 6502 CPU mit der Arbeit an der Speicherstelle FFFC, während die Z 80 CPU bei der Adresse 0000 beginnt.

Weiterhin unterscheiden sich die beiden Prozessoren in den üblicherweise verwendeten Taktfrequenzen. Diese werden häufig fälschlicherweise zum Vergleich der Arbeitsgeschwindigkeit von unterschiedlichen Prozessoren herangezogen. Ein Vergleich ist direkt nur innerhalb gleicher Prozessorfamilien möglich.

Die Prozessoren der 80er-Familie arbeiten normalerweise mit wesentlich höheren Taktfrequenzen als die Prozessoren der 65er-Familie. Die Schlußfolgerung, daß die 80er-Prozessoren damit grundsätzlich schneller sind als die 65er-Prozessoren, wäre aber falsch. Ein Mikroprozessor benötigt zur Abarbeitung eines Befehls in der Regel mehrere Taktzyklen. Vergleicht man nun Befehle des Z 80 und des 6502, die die gleiche Wirkung haben, so stellt man fest, daß der Z 80 in der Regel mehr Taktzyklen für den wirkungsgleichen Befehl benötigt als der 6502-Prozessor. Er muß, um auf die gleiche »Arbeitsgeschwindigkeit« zu kommen, mit einer höheren Taktfrequenz arbeiten.

Der Z 80-Prozessor ist trotzdem verhältnismäßig schnell, weil er eine Methode benutzt, die als Pipelining bezeichnet wird. Vereinfacht ausgedrückt, bewirkt dieses sogenannte Pipelining, daß der Prozessor, während er einen Befehl bearbeitet, schon den nächsten Befehl lädt.

Der Befehlssatz des Z 80-Mikroprozessors

Die Befehle des Z 80 lassen sich wie beim 6502 in bestimmte Befehlsgruppen einteilen.

1. Transportbefehle (auch Transferbefehle genannt)
2. Arithmetische Befehle (Addition)
3. Logische Operationen (UND-Verknüpfung)
4. Registeranweisungen (Übertragsbit löschen)
5. Sprungbefehle (Setze das Programm an der angegebenen Adresse fort)
6. Programmunterbrechungen (Unterbrechung wird zugelassen)
7. Unterprogrammbehandlung (Unterprogrammaufrufe)
8. Sonstige Befehle (NOP, Leerbefehl no operation).

Wie bekannt, lassen sich mit 8 Bit maximal 256 Werte darstellen (0-255), das heißt, der Befehlssatz eines 8-Bit-Mikroprozessors kann normalerweise maximal 256 Befehle betragen. Der 6502 besitzt 151 erlaubte (legale) Befehle (Opcodes) und in der C-MOS-Version 65C02 178 Befehle. Die restlichen Opcodes (bis zu einer Summe von 256) sind die sogenannten »illegalen Opcodes«, für deren Funktion die Hersteller keine Garantie übernehmen.

Auch der 8080 bleibt mit der Anzahl seiner Befehle (242) im Rahmen der mit 8 Bit darstellbaren Werte.

Der Z 80 hingegen besitzt mehr als 700 bekannte Befehle. Theoretisch sind es sogar mehr als 1000. Wie ist das bei einem 8-Bit-Mikroprozessor, der auch noch aufwärtskompatibel zum 8080/8085 ist, überhaupt möglich? Die Lösung ist einfacher als man glaubt. Beim Z 80 werden einige Opcodes, die beim 8080/8085 nicht benutzt werden, zur Umschaltung des gesamten Befehlssatzes verwendet. Und zwar sind es die Opcodes DD, FD, ED und CB (hexadezimale Schreibweise). Findet der Z 80 einen dieser Opcodes in einem Programm vor, schaltet er seinen Befehlssatz um. Das diesen Opcodes folgende Byte wird dann als neuer Opcode interpretiert. Diese neuen Befehle bestehen aus mindestens 2 Byte (dem »Umschaltopcode«, gefolgt von einem Befehl). Diese genial einfache Lösung zur Erweiterung des Befehlssatzes für einen 8-Bit-Prozessor scheint den Entwicklern des Z 80 so gut gefallen zu haben, daß sie diese Methode noch weiter ausgebaut haben und bei den »Umschaltopcodes« DD und FD noch weitere Umschaltopcodes vorgesehen haben. Folgt dem DD oder dem FD ein CB, wird jeweils wieder in einen neuen Befehlssatz umgeschaltet. Diese Befehle bestehen dann mindestens aus 3 Byte (3-Byte-Befehle).

Die beim Vorgänger des Z 80 (dem 8080) nicht benutzten Opcodes DD, FD, ED, CB, DDCB und FDCB schalten also beim Z 80 den gesamten Befehlssatz um. So ist es also möglich, mit einem 8-Bit Prozessor einen Befehlssatz mit mehr als 256 Befehlen zu erzeugen. Nach diesem Prinzip funktioniert auch das sogenannte »Bank-switching«, die Speicherbankumschaltung bei 8-Bit-Computern mit mehr als 64 KByte Speicher (wie auch beim C 128).

Z 80-Mnemonics

Die Firma Zilog hat für den Z 80 eine eigene Form von Mnemonics (Gedächtnishilfe für Opcodes) entwickelt, die stark von den Intel-Mnemonics abweichen. Wer sich schon mit der 6502-Maschinensprache beschäftigt hat und die 6502-Mnemonics kennt, kann sich aber freuen. Viele der Z 80-Mnemonics ähneln stark den 6502-Mnemonics.

6502-Befehl:

```
LDA $1000
```

Wirkung: Lade den Akku mit dem Inhalt der Speicherstelle an Adresse 1000 hex.

Z 80 Befehl:

```
LD A,(1000h)
```

Wirkung: Gleiche Wirkung wie beim 6502. Die Klammer um die Adresse steht bei den meisten Z 80-Assemblern für: Inhalt der Speicherstelle mit der Adresse in der Klammer in den Akku übertragen. Weiterhin erwarten die meisten Z 80-Assembler eine Angabe, in welchem Zahlensystem die Adressenangabe erfolgt. In diesem Beispiel besagt das »h« hinter der Adresse, daß es sich um eine hexadezimale Adressenangabe handelt. Wird das »h« weggelassen, so nehmen die meisten Z 80-Assembler an, daß es sich um eine Dezimalzahl handelt, und rechnen die vermeintliche Dezimalzahl in eine Hexadezimalzahl um.

Doch nun zu den Standardbefehlen der einzelnen Befehlsgruppen.

Es hat wenig Sinn, im Rahmen dieses Artikels sämtliche Z 80-Befehle (über 700) aufzulisten und in ihrer Wirkung zu beschreiben. Selbst im Rahmen eines Z 80-Lehrbuches wäre diese Methode nicht nur langweilig, sondern auch unsinnig, da viele Befehle im Prinzip die gleiche Wirkung haben, sich jedoch nur auf andere Register beziehen. Deshalb werden hier die Standardbefehle der einzelnen Befehls-

gruppen besprochen. Dabei wird das Prinzip dieser Befehle deutlich, das allen anderen der gleichen Kategorie zugrunde liegt, um den Anwender (beziehungsweise den Lernenden) in die Lage zu versetzen, alle Befehle dieser Art bei Bedarf einzusetzen. Um größere Programme zu erstellen, sollte man sich dann einen kompletten Befehlssatz des Z 80 besorgen und die geeignetesten Befehle einsetzen.

Die für den Einstieg in die Z 80-Programmierung wichtigen Befehlsgruppen werden im folgenden ausführlicher behandelt.

Transferbefehle

Mit Hilfe von Transferbefehlen können:

- Daten vom Prozessor (aus einem seiner Register) zum Speicher (RAM) und umgekehrt vom Speicher (RAM oder ROM) zum Prozessor
- Daten vom Prozessor zu einem Ein-/Ausgabebaustein und umgekehrt
- Daten innerhalb des Prozessors von einem Register in ein anderes Register übertragen werden.

Diese Befehle werden in fast jedem Z 80-Maschinen- oder Assemblerprogramm benötigt.

Die erste Kategorie der Transferbefehle sind die sogenannten Ladebefehle. Jeder Ladebefehl beginnt mit einem »LD« und einem Leerzeichen. Danach erfolgt die Angabe des Ziels und die Quelle der Daten. Diese beiden Angaben müssen durch ein Komma getrennt werden. Geladen werden kann jedes Register (Ziel) mit dem Inhalt jedes anderen Registers (Quelle) oder einer beliebigen Speicherstelle (Quelle). Es können sogar Register mit ihrem eigenen Inhalt geladen werden, wenn bei Ziel und Quelle die gleiche Registerangabe erfolgt. Die Wirkung eines solchen scheinbar unsinnigen Befehls ist eine kleine Zeitverzögerung des ablaufenden Programms. Diese Verzögerung kann manchmal aber recht nützlich sein (zum Beispiel bei Steuerungen). Die Aussage, daß jedes Register wie oben angegeben geladen werden kann, trifft natürlich auch auf den Akkumulator zu. Jedoch besitzt er, wie beim 6502, eine Sonderstellung. Das weiter oben angegebene Beispiel (Lade Akku 6502 und Z 80) ist so ein Sonderbefehl für den Akku. Doch zunächst einige Beispiele für normale Transferbefehle:

Lade ein beliebiges Register mit einem angegebenen Wert (unmittelbare Adressierung)

`LD A,1FH` – Bedeutung: Lade den Akkumulator mit dem Wert 1F hexadezimal
`LD B,2AH` – Lade das B-Register mit dem Wert 2A hex.
`LD H,05H` – Lade das H-Register mit dem Wert 5 hex.

Jedes der sogenannten »Arbeitsregister« (A,B,C,D,E,H und L-Register) kann auf diese Weise mit einem beliebigen Wert zwischen 00 und FF (hexadezimal) geladen werden.

Doch nun zum zweiten Beispiel:
Lade ein Register mit dem Inhalt eines anderen Registers.
`LD A,B` – Lade Akkumulator mit dem Inhalt des Registers B.
`LD B,C` – Lade das Register B mit dem Inhalt des Registers C.
`LD H,A` – Lade das Register H mit dem Inhalt des Akkus.

Bereits diese wenigen Beispiele machen das Prinzip deutlich. In gleicher Weise kann aus jedem Arbeitsregister in ein anderes geladen werden. Bei diesen Befehlen wird der Akku wie ein »normales« Register behandelt.

Wie bereits angesprochen, existieren auch Befehle, um die Werte beliebiger Speicherstellen in ein Register zu bringen. Diese Befehle sind nicht sofort durchführbar. Bevor so eine Operation durchgeführt werden kann, muß die Adresse der Speicherstelle in eines der Registerpaare geladen werden.

An dieser Stelle kommen wir erstmals auf die weiter oben bereits erwähnten Registerpaare zu sprechen. Die Kombination aus B- und C-Register wird als BC-Register (16 Bit), die Kombination von D- und E-Register als DE-Register (16 Bit) und die Kombination aus H- und L-Register als HL-Register (16 Bit) bezeichnet. Somit stehen drei »16-Bit-Register« (Arbeitsregister) zur Aufnahme von 16-Bit-Werten, wie zum Beispiel Adressen, zur Verfügung.

Es gibt jetzt zwei Möglichkeiten, diese »16-Bit-Register« mit Werten zu laden. Die erste besteht darin, nacheinander beide Register einzeln mit den beiden Bytes einer Adresse zu laden. Die zweite Möglichkeit besteht in der Verwendung von sogenannten »16-Bit-Befehlen«, mit denen eines der Registerpaare durch einen einzigen Befehl geladen wird.

Soll nun ein beliebiges Register mit dem Inhalt einer Speicherstelle geladen werden, so muß die Adresse der Speicherstelle in einem der Registerpaare stehen. Die Befehlsfolge könnte folgendermaßen aussehen:

`LD H,10H` – Lade das H-Register mit dem Hex-Wert 10
`LD L,00H` – Lade das L-Register mit dem Hex-Wert 00
`LD C,(HL)` – Lade das C-Register mit dem Inhalt der Speicherstelle, deren Adresse im HL-Register steht. In unserem Beispiel wäre das die Adresse 1000 hexadezimal.

Zu beachten ist hierbei, daß das höherwertige Byte der Adresse im H-Register und das niederwertige Byte der Adresse im L-Register stehen muß.

Sehen wir uns noch eine zweite Möglichkeit zum Laden von »16-Bit-Register« an:

`LD HL,1000H` – Lade das Registerpaar HL mit dem Wert 1000 hex.
`LD C,(HL)` – Die Wirkung ist äquivalent zu obigem Beispiel.

Der Befehl »LD HL,nnnn« ist ein sogenannter 16-Bit-Befehl, das heißt, mit einem einzigen Befehl kann ein 16-Bit-Wert in ein Registerpaar geladen werden. Diese Möglichkeit, die auf alle Registerpaare anwendbar ist (auch auf Programmzähler und Indexregister), erleichtert die Programmierarbeit, spart Zeit und Speicherplatz. Nicht mehr, aber auch nicht weniger leisten diese oft genannten »16-Bit-Befehle« des »8-Bit-Prozessors« Z 80.

Wie schon weiter oben erwähnt, nimmt der Akkumulator unter den Registern eine Sonderstellung ein. So auch beim Laden mit dem Inhalt einer Speicherstelle. Die soeben beschriebenen Ladebefehle sind die allgemeine Form für alle Register. Der Akkumulator kann auch direkt mit dem Inhalt einer beliebigen Speicherstelle geladen werden. Dazu wird der folgende Befehl verwendet:

`LD A,(1000H)` – Lade den Akku mit dem Inhalt der Speicherstelle an Adresse 1000 hex.

Bevor wir weitere Befehle (Befehlsgruppen) des Z 80 kennenlernen, sei auf eine Besonderheit der meisten Z80-Assembler hingewiesen. Versucht man zum Beispiel mit dem Befehl »LD A,(A000h)« den Inhalt der Speicherstelle A000 (hex) in den Akkumulator zu laden, streiken die meisten Z 80-Assembler. Sie erwarten bei Zahlenangaben (Adressen) als erste Ziffer eine Zahl zwischen 0 und 9. Dieses Problem läßt sich auf denkbar einfache Weise lösen. Man fügt zur Zahlenangabe einfach eine führende 0 hinzu.

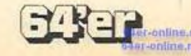
Der Befehl »LD A,(0A000h)« wird dann anstandslos übersetzt.

Testen Sie Ihren Z 80-Assembler in dieser Beziehung, bevor Ihnen unnötigerweise graue Haare wachsen.

Arithmetische Befehle

Der Z 80 kann, wie die meisten Mikroprozessoren, grundsätzlich nur addieren und subtrahieren. Da er diese Operationen jedoch sehr schnell ausführt, können andere (Multiplikation oder Division) durch kleine Programme nachgebildet werden. Die Multiplikation 3*4 kann auch als Addition 4+4+4 aufgefaßt und so programmiert werden.

Unterschieden wird bei den arithmetischen Befehlen zwischen Operationen mit Übertrag und ohne Übertrag. Weiter-



hin ist zu berücksichtigen, daß die meisten Additions- und Subtraktionsbefehle über den Akkumulator laufen. Additionen ohne Berücksichtigung des Übertrags sehen folgendermaßen aus:

`ADD A,03h` – Addiere zum Akkuinhalt den Wert 03 hex. Das Ergebnis steht im Akkumulator.

`ADD A,B` – Addiere zum Akkuinhalt den Inhalt des B-Registers. Das Ergebnis steht im Akku.

`ADD A,(HL)` – Addiere zum Akkuinhalt den Inhalt der Speicherstelle, deren Adresse im HL-Register steht. Das Ergebnis steht im Akku.

Beim letzten Additionsbefehl muß, bevor addiert werden kann, das HL-Register mit der Adresse der Speicherstelle geladen werden (siehe Transferbefehle). Die Doppelregister erlauben es, mit dem Z 80 16-Bit-Arithmetik zu programmieren:

`ADD HL,BC` – Addiere zum Inhalt des HL-Registers (16 Bit) den Inhalt des BC-Registers (16 Bit). Das Ergebnis steht im HL-Register.

`SUB 05H` – Subtrahiere vom Akkuinhalt den Wert 5 hex. Das Ergebnis steht im Akkumulator.

`SUB C` – Subtrahiere vom Akkuinhalt den Inhalt des C-Registers. Das Ergebnis steht im Akku.

Soll ein eventuell entstandener Übertrag berücksichtigt werden, programmiert man mit folgenden Befehlen:

`ADC A,09H` – Addiere zum Akkumulatorinhalt die »Hexzahl« 09 und das Übertragsflag C (C = Carryflag). Das Ergebnis steht im Akku.

`ADC A,C` – Addiere zum Akkumulatorinhalt den Inhalt des C-Registers und das Übertragsflag.

Die entsprechenden 16-Bit-Befehle lauten:

`ADC HL,DE` – Addiere zum Inhalt des HL-Registers den Inhalt des DE-Registers und das Übertragsflag C. Das Ergebnis steht im HL-Register.

`SBC A,04H` – Subtrahiere vom Akkuinhalt den Wert 04 hex sowie das Übertragsflag C. Das Ergebnis steht im Akkumulator.

Indizieren mit Hilfe von Registern

Alle Register und Registerpaare können »inkrementiert« oder »dekrementiert« werden. Dabei ist zu beachten, daß die folgenden Befehle, auf Registerpaare angewendet (16 Bit), keine Flags im Statusregister beeinflussen.

```
INC A  ;erhöhe den Inhalt des Akkus um 1
INC B  ;erhöhe den Inhalt des B-Registers um 1
INC HL ;erhöhe den Inhalt des HL-Registerpaares
        um 1
DEC A  ;verringere den Inhalt des Akkus um 1
DEC HL ;verringere den Inhalt des HL-Registers um 1
```

Diese Befehle können für Zählaufgaben oder Zeitschleifen eingesetzt werden.

Weitere arithmetische Befehle finden Sie in der folgenden Tabelle.

DAA	Dezimalanpassung (BCD-Code)
CPL	Bildung des Einerkomplements des Akkus
NEG	Bildung des Zweierkomplements des Akkus

Der Z 80 besitzt die Logikbefehle AND (Bitweise UND-Verknüpfung zweier Bytes), OR (Bitweise ODER-Verknüpfung zweier Bytes), XOR (EXKLUSIV-ODER-Verknüpfung zweier Bytes) sowie den Vergleichsbefehl CP ("C"om"p"are).

`AND 83H` – Führt eine UND-Verknüpfung jedes Bits des Akkus mit jedem Bit der Zahl 83 hex durch. Für den Akkuinhalt OF hex würde dieser Befehl folgendes bewirken:

Akkuinhalt	OF hex =	00001111 binär
	83 hex =	10000011 binär
AND-Befehl	ergibt :	00000011 binär im Akku

Nach dem AND-Befehl würde also 03 im Akku stehen. Mit Hilfe des AND-Befehls, der auf keinen Fall mit einer Addition verwechselt werden darf, können beliebige Bits »ausgeblendet« werden. In diesem Zusammenhang spricht man auch von »Maskierung«. Diese Maskierung ist bei Ein-/Ausgabeoperationen häufig sehr hilfreich.

Verknüpfungsbefehle des Z 80:

AND C	Bitweise UND-Verknüpfung der Inhalte des Akkus und des C-Registers. Ergebnis steht im Akku.
OR 1FH	Bitweise ODER-Verknüpfung des Akkuinhalts mit dem »Datenbyte« 1F hex. Das Ergebnis steht im Akku. Die Verknüpfung wird, wie beim AND-Befehl beschrieben, durchgeführt, mit dem Unterschied, daß, sobald ein Bit im Akku oder im »Datenbyte« oder in beiden den Wert »1« besitzt, das Ergebnis 1 ist.
OR E	Bitweise ODER-Verknüpfung des Akkuinhalts mit dem Inhalt des E-Registers. Das Ergebnis steht im Akku.
XOR 1AH	Bitweise EXCLUSIV-ODER-Verknüpfung des Akkuinhalts mit dem »Datenbyte« 1A hex. Das Ergebnis steht im Akku. Die EXCLUSIV-ODER-Verknüpfung liefert als Ergebnis immer dann eine »1«, wenn die Inhalte der einzelnen Bits unterschiedlich sind. Sind sie gleich, entsteht eine »0«. Dieser Befehl kann zur sogenannten Invertierung (Umkehrung) benutzt werden (bei Logikanpassungen).
CP 1BH	Vergleiche den Akkuinhalt mit dem »Datenbyte« 1B hex. Dabei wird 1Bh vom Akkuinhalt subtrahiert und das Ergebnis nicht weiter berücksichtigt. Es werden lediglich Flags im Statusregister beeinflußt.
CP D	Vergleiche den Akkuinhalt mit dem Inhalt des D-Registers. Die Wirkung ist die gleiche wie bei vorherigem Befehl.

Alle logischen Operationen des Z 80 arbeiten nur mit 8-Bit-Daten.

Mit Hilfe der sogenannten Registeranweisungen kann der Inhalt eines Registers verschoben werden. Der Inhalt jedes Bits wird in ein benachbartes Bit verschoben. Beim sogenannten Rotieren wird ein »herausfallendes« Bit jeweils am Anfang oder Ende des Registers verschoben. Beim Z 80 kann das Carryflag bei diesen Operationen mit einbezogen werden.

Auch hier wieder ein Beispiel:

`RRA` ;lasse den Inhalt nach rechts rotieren, das heißt jedes Bit (Inhalt) wird um 1 Bit nach rechts verschoben, der Inhalt von Bit 0 gelangt ins Carryflag, der Inhalt des Carryflags in Bit 7.

Weiterhin können mit Hilfe von Registeranweisungen einzelne Bits im Flagregister beeinflußt werden.

Der Z 80 ist im Gegensatz zu den meisten anderen Mikroprozessoren mit einer Vielzahl von Befehlen zur Bitmanipulation ausgestattet, deren Beschreibung hier aber zu weit führen würde.

Sprungbefehle

Sprungbefehle werden in fast jedem Maschinenprogramm benötigt. Wie beim 6502 (und anderen Mikroprozessoren) kann ein Sprung, das heißt eine Fortsetzung des Programm an einer angegebenen Adresse, von bestimmten Bedingungen abhängig gemacht werden. Ein Sprung ohne zusätzliche Bedingung wird unbedingter Sprung, ein Sprung, der nur durchgeführt wird, wenn eine zusätzliche Bedingung erfüllt ist, wird bedingter Sprung genannt.

Beispiele für unbedingte Sprünge:

JP 1000h setze das Programm mit dem Befehl in der Speicherstelle 1000h fort.

JP (HL) setze das Programm an der Adresse fort, die im HL-Register steht.

Die Bedingung, die bei bedingten Sprüngen erfüllt sein muß, damit der jeweilige Sprung durchgeführt wird, ist der Zustand eines entsprechenden Flags im Flag-Register. Es kann der Zustand (0 oder 1) des Z, C, P/V oder S-Flags sein.

Beispiele für bedingte Sprünge:

JP Z,4000h setze das Programm an der Adresse 4000h fort, wenn das Zero-Flag 0 ist.

JP NZ,5000h setze das Programm an der Adresse 5000h fort, wenn das Zero-Flag nicht 0, also 1 ist.

Das Zero-Flag wird auf »1« gesetzt, wenn bei einer arithmetischen oder logischen Operation vorher, ein Ergebnis »0« geworden ist.

Beim Z 80 kann eine Programmunterbrechung (Interrupt) durch einen Baustein außerhalb des Prozessors durch entsprechende Befehle zugelassen oder unterbunden werden (maskierbare Interrupts). Hierfür dient der INT-Anschluß am Prozessor.

Manipulation der Interrupts:

EI setze das Interrupt-Flip-Flop. Danach sind Interrupts zugelassen.

DI setze das Interrupt-Flip-Flop zurück. Danach ist ein Interrupt über den INT-Anschluß nicht mehr möglich.

Zusätzlich besitzt der Z 80-Mikroprozessor noch eine »nicht maskierbare Interruptleitung« (NMI-Anschluß). Eine Interrupt-Anforderung über diese Leitung wird auf jeden Fall durchgeführt. Weiterhin gibt es beim Z 80 noch drei Interruptmodi, die den Ablauf nach einem Interrupt festlegen (Befehle: IM 0, IM 1 und IM 2). Damit der Prozessor bei Unterbrechungen nicht durcheinanderkommt, wird bei einem Interrupt der gerade bearbeitete Befehl zu Ende bearbeitet, bevor der Prozessor auf einen Interrupt reagiert.

Unterprogrammbehandlung

Unterprogramme können beim Z 80 mit Hilfe der CALL-Befehle aufgerufen werden. Der CALL-Befehl bewirkt einen Sprung zu einem Unterprogramm an einer angegebenen Startadresse. Wie die Sprungbefehle, können auch die Unterprogrammaufrufe an Bedingungen geknüpft werden. Im Gegensatz zu den Jump-Befehlen (JP) »merkt« sich der Prozessor jedoch seine »Absprungadresse«. Er kann nach der Beendigung des Unterprogramms das ursprüngliche Programm mit dem Befehl fortsetzen, der dem CALL-Befehl im Hauptprogramm folgt. Dieser Rücksprung erfolgt, wenn der Prozessor im Unterprogramm einen RET-Befehl (RETURN) vorfindet, das heißt zu jedem CALL-Befehl im Hauptprogramm gehört ein RET-Befehl im Unterprogramm. Auch hierzu wieder ein Beispiel:

Hauptprogramm:

beliebiger Befehl

CALL 2000H setze das Programm an der Adresse 2000 hex. fort.

nnnn nächster Befehl

Unterprogramm:

2000 beliebiger Befehl

RET springe zurück ins Hauptprogramm zum nächsten Befehl (Adresse nnnn)

Es können nicht nur eigene Unterprogramme aufgerufen werden, sondern auch Unterprogramme des Betriebssystems (CP/M-BDOS-Routinen). Diese Möglichkeit erleichtert die Programmierarbeit mit dem Z 80 ganz enorm. In den Programmbeispielen machen wir von dieser Möglichkeit Gebrauch.

Sonstige Befehle

In diese Gruppe sind die Befehle eingeordnet, die in keine der anderen Gruppen passen. Zu diesen Befehlen gehört der Befehl »HALT«. Die CPU unterbricht die Programmausführung und führt so lange den Befehl »no operation« (NOP) aus, bis ein Reset durchgeführt wird oder ein Interrupt erfolgt. Der Befehl NOP gehört in diese Gruppe. Er bewirkt nur eine kleine Zeitverzögerung (1 Maschinenzyklus) des ablaufenden Programms. NOPs können in ein Programm genommen werden, um das Programm später an dieser Stelle mit zusätzlichen Befehlen zu erweitern.

Zum Abschluß sollen noch zwei kleine Beispielprogramme (Listing 1 und Listing 2) vorgestellt werden, die für einen Einstieg in die praktische Programmierarbeit benutzt werden können. Um diese Programme auszuprobieren, benötigt man einen Editor (ED, Wordstar oder etwas ähnliches), einen Z 80-Assembler, der das mit dem Editor erstellte Assemblerprogramm in den Maschinen-Code übersetzt und einen Linker, der aus dem übersetzten Programm eine direkt unter CP/M aufrufbare »COM«-Datei erstellt.

Die Namenserweiterung muß nach den Erfordernissen des verwendeten Assemblers gewählt werden. Der bei der Erstellung dieser Beispielprogramme verwendete Makroassembler des Small-C-Entwicklungssystems verlangt die Erweiterung MAC, andere Assembler erwarten den Zusatz ASM. Ganze Zeilen werden vom Assembler als Kommentarzeilen angesehen, wenn als erstes Zeichen ein »;« steht.

Die Bedienung Ihres Editors, Assemblers und Linkers entnehmen Sie bitte den Handbüchern zu diesen Programmen.

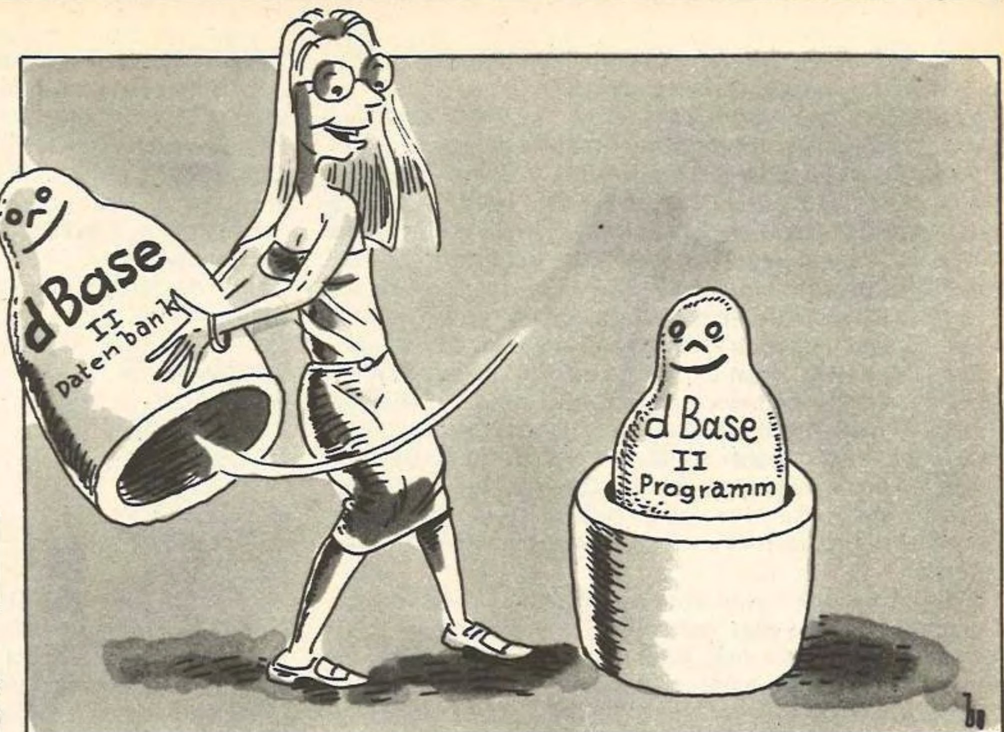
(Rüdiger Szillus/rf)

```
; Testprogramm Z 80 (TEST.MAC)
; Ausgabe eines Zeichens auf dem Bildschirm
;
ORG 0100H     ;Startadresse des Programms
LD C,02H      ;Lade das C-Reg. mit dem Wert 02 hex(BDOS-
              ;Funktion 2)
LD E,"R"      ;Lade das E-Register mit dem ASCII-Wert
              ;für R
CALL 5        ;Sprung ins CP/M-BDOS, Aufruf der BDOS-
              ;Funktion 2
CALL 0        ;Rücksprung zu CP/M
END           ;Ende des Quellprogramms
;
;Beim Sprung ins BDOS wird die BDOS-Funktion
;ausgeführt, deren Nummer im C-Register steht (hier
;BDOS-Funktion 2)
;Die BDOS-Funktion 2 gibt das ASCII-Zeichen auf dem
;Bildschirm
;aus, dessen Wert im E-Register steht (hier R).
```

Listing 1. Ausgabe eines einzelnen Zeichens

```
; 2. Testprogramm (TEST2.MAC)
; Ausgabe eines Strings auf dem Bildschirm
;
      ORG 0100h                 ;Startadresse 100 hex.
      LD C,09h                  ;Lade den Wert 09 hex. ins
                                ;C-Register
      LD DE,TEXT                ;Lade die Anfangsadresse
                                ;von "TEXT:"
      CALL 5                    ;Sprung ins BDOS
      CALL 0                    ;Rücksprung zu CP/M
;
TEXT: DB "Rüdiger Szillus $";Definition des Datenbytes
      END                       ;Ende des Quellprogramms
;
;Beim Sprung ins BDOS wird die BDOS-Funktion 9
;ausgeführt.
;(Nummer steht im C-Register).
;Die Adresse des auszugebenden Strings steht im DE-
;Register ;(hier Label TEXT:).
;BDOS-Funktion 9 = Ausgabe eines Strings auf dem
;Bildschirm.
;Das $-Zeichen markiert das Ende des Strings und muss
;angegeben werden.
```

Listing 2. Ausgabe eines Strings



dBase II als Programmiersprache

Um alle Möglichkeiten, die dBase II bietet, ausschöpfen zu können, benötigt man Kenntnisse über die Programmierung dieser leistungsfähigen Dateiverwaltung. Wir zeigen Ihnen, wie Sie die dBase-II-Datenbanksprache für eigene Anwendungen heranziehen können und somit ein Programm im Programm benutzen.

Spricht man über CP/M, tauchen immer wieder die Namen von drei Programmen auf: Wordstar 3.0, Multiplan 128 und dBase II. Jeder weiß dann zu berichten, daß die einzelnen Programme in der Reihenfolge ihrer Nennung der Textverarbeitung, der Tabellenkalkulation und der Dateiverwaltung dienen. Dies stimmt natürlich, jedoch stellt man mit dieser pauschalen Aussage das Licht von dBase II unter den Scheffel. dBase II ist wohl das bekannteste Datenbanksystem für den professionellen Einsatz und außerdem eine sehr leistungsstarke Programmiersprache, die bestimmte Elemente von Basic und Pascal beinhaltet. Hinzu kommen spezielle Befehle für die Bearbeitung von Dateien, die in dieser Form und Häufung nicht zu übertreffen sind (ausgenommen die neue Version dBase III für IBM und kompatible Computer).

Dieser Bericht soll am Beispiel einer einfachen modular aufgebauten Adressenverwaltung in die Benutzung von dBase II als Programmiersprache einführen und dem interessierten Leser die Möglichkeit der unbegrenzten Erweiterung und Änderung des Programms bieten.

Die Vorbereitung

Bevor wir anfangen zu arbeiten, müssen wir eine entsprechende Arbeitsdiskette vorbereiten. Dazu gehen Sie folgendermaßen vor:

1. Schalten Sie das Diskettenlaufwerk ein
2. Legen Sie die CP/M-Systemdiskette in das Laufwerk
3. Schalten Sie den Computer ein
4. Nach Erscheinen des A> geben Sie »PIP« ein und drücken <RETURN>
5. Nach Erscheinen des »*« entfernen Sie die CP/M-Diskette und legen die dBase-II-Diskette Nr. 1 ein
6. Tippen Sie nun »e:=a:*.*« ein und drücken Sie <RETURN>
7. Nach mehreren Diskettenwechseln, zu denen Sie Ihr Computer jeweils auffordert, besitzen Sie eine Arbeitskopie von dBase II
8. Wenn wieder »*« erscheint, drücken Sie <RETURN>, wodurch Sie wieder ins CP/M-Betriebssystem gelangen, was Sie leicht am A> erkennen können
9. Legen Sie die Arbeitskopie von dBase II ins Diskettenlaufwerk
10. Geben Sie »dBase II« ein und drücken Sie <RETURN>
11. Beantworten Sie die Frage nach dem Datum und drücken Sie <RETURN>.

Sie befinden sich nun in dBase II, das auch eine Art Interpreter-Hochsprache darstellt und sich jeweils mit einem ».« meldet. Bevor wir in die eigentliche Arbeit mit dieser Sprache einsteigen, müssen Sie die Struktur der Datenbank vorbereiten. Geben Sie dazu ein:

CREATE adressen <RETURN>

Nach einer kurzen Weile stellt dBase II fest, daß es sich hierbei um eine neue Datei handelt und will nun von Ihnen wissen, welche Struktur die Datei besitzen soll. Geben Sie die folgenden Daten ein:

```
ANREDE,    c,   5   <RETURN>
NAME,      c,  20   <RETURN>
VORNAME,   c,  20   <RETURN>
STRASSE,   c,  20   <RETURN>
PLZORT,    c,  20   <RETURN>
TELEFON,   c,  15   <RETURN>
GEBDAT,    c,   8   <RETURN>
NOTIZ,     c,  40   <RETURN>
<RETURN>
```

Auf die Frage, ob Sie gleich Daten eingeben wollen, antwor-

ten Sie mit »N« für »Nein«. Benutzen Sie den Befehl USE <RETURN>, um die Datei zu schließen.

Wir wollen nun kurz untersuchen, was Sie eigentlich gemacht haben. Sie haben dBase II mitgeteilt, wie die einzelnen Felder Ihres Datensatzes heißen, nämlich ANREDE, NAME, VORNAME, STRASSE, PLZORT, TELEFON, GEBDAT und NOTIZ. Durch die Eingabe von »C« nach dem Feldnamen weiß dBase II, daß in diesen Feldern später alphanumerische Zeichen, das heißt Buchstaben, Zahlen und Satzzeichen stehen können. Die jeweiligen Zahlen sagen dBase II, wieviel Zeichen es für das jeweilige Feld reservieren soll.

Sie können die Feldnamen frei wählen, jedoch müssen diese immer mit einem Buchstaben beginnen. Es ist sinnvoll, Feldnamen zu wählen, die auch ohne nähere Erklärung aussagekräftig sind. Feldnamen dürfen aus bis zu 10 Zeichen bestehen und ein Feld kann bis zu 254 Zeichen beinhalten. Das Limit für die Anzahl der Felder beträgt 32, jedoch darf die Gesamtlänge eines Datensatzes 1000 Zeichen nicht überschreiten.

Außer »C« kennt dBase II noch die Feldtypen »N« und »L«. »N« geben Sie ein, wenn es sich bei dem Datenfeld um ein numerisches Feld handelt, also um ein Feld mit Zahlen, mit dem Sie später eventuell rechnen wollen. In diesem Fall können Sie ein weiteres Komma und danach die Zahl der Stellen nach dem Komma eingeben. »L« bezeichnet logische Felder, die nur »TRUE« (stimmt) oder »FALSE« (stimmt nicht) sein können. Ein solches Feld könnte zum Beispiel MITGLIED sein, da hier nur die Antwort »Ja« oder »Nein« sinnvoll wäre. Logische Felder haben immer die Länge »1«.

Beachten Sie, daß das Feld GEBDAT, in dem später das Geburtsdatum stehen soll, acht Zeichen lang ist. Merken Sie sich, daß bei Zahlen und Daten der Punkt und das Komma mitgezählt werden (tt/mm/jj = acht Zeichen).

Nun müssen wir uns überlegen, was unser Programm können soll:

1. Es soll menügesteuert und
2. jederzeit erweiterbar sein

Weiterhin sollte der Anwender folgende Möglichkeiten zur Verfügung haben:

3. Neue Datensätze eingeben
4. Datensätze löschen
5. Datensätze ändern
6. auf dem Bildschirm in der Datei blättern
7. Adreßlisten und Etiketten drucken

Bestimmt vermissen Sie hier eine Möglichkeit: die des Sortierens. Diese brauchen wir jedoch nicht als eigenen Programmpunkt zu integrieren, da uns dBase II hierbei zu Hilfe kommt, indem es die Möglichkeit der Indizierung bietet. Bei der Indizierung wird die in der Regel unsortierte Hauptdatei nach einem oder mehreren Feldern sortiert. Dabei wird eine zweite Datei, die Indexdatei, angelegt. Die Hauptdatei bleibt jedoch unsortiert. Mit Hilfe dieser Indexdatei kann sehr schnell auf beliebige Daten zugegriffen werden.

Wir indizieren nun unsere Datei, obwohl noch kein einziger Datensatz darin ist. Bei entsprechender Programmierung wird dBase II nun jeden neuen Datensatz richtig einordnen. Geben Sie folgendes ein:

USE ADRESSEN <RETURN>
INDEX ON NAME+VORNAME TO NAME <RETURN>
USE <RETURN>

Später wird dann dBase II die Familiennamen in alphabetischer Reihenfolge ordnen, bei gleichem Familiennamen zusätzlich noch nach Vornamen.

Sehen wir uns nun mal an, welche Files auf unserer Diskette zu finden sind. Neben den dBase-II-Dateien von der Programmdiskette finden wir, nachdem wir

LIST FILES LIKE *.* <RETURN>

eingegeben haben, die folgenden Einträge:
ADRESSEN.DBF, unsere noch leere Datei und
NAME.NDX, unsere Indexdatei.

Wir wollen uns, bevor es richtig losgeht, auch noch einmal unsere Datei ansehen, indem wir eingeben:

USE ADRESSEN <RETURN>
LIST STRUCTURE <RETURN>

Folgendes Bild sollte erscheinen:

Strukturdaten für Datei: A:ADRESSEN. DBF
Anzahl der Sätze: 00000
Datum der letzten Aktualisierung: 00/00/00
Primäre Datei

Feld Name	Typ	Länge Dez. St.
001 ANREDE	»C«	005
002 NAME	»C«	020
003 VORNAME	»C«	020
004 STRASSE	»C«	020
005 PLZORT	»C«	020
006 TELEFON	»C«	015
007 GEBDAT	»C«	008
008 NOTIZ	»C«	040
** Gesammt **		00149

Nebenbei bemerkt, lassen Sie sich von den wenigen Rechtschreibfehlern von dBase II nicht stören, seine Arbeit verrichtet es dennoch fehlerfrei.

Nachdem wir nun alles verglichen und keinen Fehler gefunden haben, gehen wir über zur eigentlichen Programmierung.

Vorbemerkung

dBase II besitzt einen eigenen Texteditor zur Erstellung von Programmen. Dieser ist jedoch vom Speicherplatz (ungefähr fünf KByte) und von den Textmanipulationsmöglichkeiten her eingeschränkt. Der Vorteil des CP/M-Betriebssystems ist, daß Sie mit jedem anderen CP/M-Textverarbeitungsprogramm dBase-II-Programme erstellen können. Dafür allein lohnt sich schon die Anschaffung von Wordstar 3.0 oder auch Turbo Pascal, das ebenfalls einen Wordstar-ähnlichen komfortablen Editor besitzt.

Die Module unseres Adreßverwaltungsprogramms sind jedoch so kurz gehalten, daß der Texteditor von dBase II auf jeden Fall ausreicht, wenn Sie die Zeilen, die mit »*« beginnen, Leerzeilen und Leerstellen am Anfang von Zeilen weglassen.

Den Texteditor von dBase II aktivieren wir durch den Befehl

MODIFY COMMAND programmname <RETURN>

wobei »programmname« für jeden von Ihnen ausgesuchten Namen stehen kann, sofern dieser eine Länge von acht Zeichen nicht überschreitet. dBase II ergänzt später beim Speichern diesen Namen automatisch durch ».CMD«.

Im Texteditor brauchen Sie folgende Tastenkombinationen:

<CONTROL+V> = Einfügemodus aus-/einschalten
<CONTROL+T> = löscht Zeile, in der der Cursor steht
<CONTROL+Y> = löscht Inhalt der Zeile, in der der Cursor steht
<CONTROL+N> = fügt Leerzeile an Cursorposition ein
<CONTROL+C> = springt halben Bildschirm weiter
<CONTROL+Q> = verläßt den Texteditor, ohne die durchgeführten Änderungen zu speichern
<CONTROL+W> = verläßt den Texteditor, speichert den geänderten Text und speichert ursprünglichen Text mit der Bezeichnung »programmname.BAK« auf Diskette.

Statt <CONTROL+W> können Sie auch die kleine Cursor-Taste direkt neben der rechten <SHIFT>-Taste drücken.

Laut Handbuch soll es möglich sein, durch Eingabe von <CONTROL+R> im Programmtext jeweils um einen halben Bildschirm zurückzugehen, jedoch weigerte sich der Computer beharrlich, diesem Befehl zu folgen und bewegte den Cursor jeweils nur um eine Zeile zurück.

Innerhalb des Textes können Sie sich mit Hilfe der vier Cursor-Tasten frei bewegen.

Nun jedoch endlich zur Erstellung unseres Programms. Nach Eingabe des Befehls

MODIFY COMMAND menu

befinden Sie sich im Texteditor von dBase II. Die erste Zeile Ihres noch nicht existenten Programms (Listing 1) ist revers dargestellt. Beginnen Sie nun, die Befehle einzugeben, die dBase II nach der Eingabe von DO MENU <RETURN> ausführen soll. Die Bedeutung der einzelnen Befehle wird jeweils erklärt werden. Lassen Sie, wie vorher schon einmal erwähnt, Zeilen, die mit einem »*« beginnen und Leerzeilen, sowie Leerzeichen am Anfang der Zeilen, weg.

SET DELETED ON sorgt dafür, daß im weiteren Verlauf des Programms zur Löschung vorgemerkte Datensätze von dBase II ignoriert werden.

SET TALK OFF unterdrückt Systemmeldungen, da diese den Bildschirmaufbau empfindlich stören können.

STORE " " TO ANTWORT

Innerhalb des Menüs wird dBase II später von uns eine Antwort erwarten, die ein Zeichen lang sein wird. Deswegen teilen wir dies dBase II mit, damit es eine Variable mit dem Namen »antwort« vorbereitet, in die später unsere aktuelle Antwort eingelesen wird.

DO WHILE ... .OR. ...

Die folgenden Programmschritte soll dBase II solange wiederholen, bis der Benutzer eine sinnvolle Antwort eingegeben hat. Da wir insgesamt sechs Menüpunkte haben, die mit den Zahlen von 0 bis 5 bezeichnet werden, soll es bei jeder anderen Eingabe in der Form reagieren, daß es wieder die Menümaske zeigt. Die Zahlen von 0 bis 5 besitzen ASCII-Codes, die direkt aufeinander folgen, also muß jedes andere eingegebene Zeichen einen ASCII-Code besitzen, der entweder niedriger oder höher ist. Wenn also ein solches Zeichen eingegeben wird, springt dBase II ständig zur DO-WHILE-Schleife zurück. Beachten Sie bitte, daß Befehle wie .OR., .AND. oder .NOT. jeweils mit einem Punkt beginnen und auch abgeschlossen werden müssen.

ERASE

Als nächstes soll der Bildschirm gelöscht werden, da wir natürlich unser Menü wie einen Brief auf einem sauberen Blatt Papier auf einem leeren Bildschirm zeigen wollen.

@ Zeile, Spalte SAY "Text"

Zwar gehört zum Lieferumfang von dBase II das Hilfsprogramm ZIP, das beim Erstellen von Bildschirmmasken nützlich ist, jedoch besitzt dieses nach Meinung des Autors den Nachteil, daß dann später im Programm jeweils die mit Zip erstellte Maske geladen werden muß, was das Programm in seiner Geschwindigkeit unnötig verlangsamt. Außerdem ist es einfacher, direkt mittels Koordinaten (im Beispiel: Zeile, Spalte) anzugeben, an denen auf dem Bildschirm bestimmte Mitteilungen erscheinen sollen. An der Stelle von Zeile geben Sie die Nummer der Bildschirmzeile ein (0 bis 24), für Spalte die entsprechende Nummer der Bildschirmspalte (0 bis 79). Zwischen den Anführungszeichen steht dann Ihr gewünschter Ausgabetext.

@ Zeile, Spalte SAY "Text" GET antwort

Dies entspricht weitgehend dem vorherigen Befehl, jedoch ist nun der Befehl GET dazugekommen. An dieser Stelle erwartet dBase II von Ihnen eine Eingabe. Diese Eingabe wird in der Variablen »antwort« gespeichert. Der Befehl READ bewirkt, daß Ihre Eingabe übernommen wird. Stellt dBase II nun fest, daß Ihre Eingabe der DO-WHILE-Bedingung entspricht, also nicht eine der Zahlen von 0 bis 5 ist, springt es zum Anfang der Schleife zurück, löscht wieder den Bildschirm, erwartet wieder eine Eingabe etc., bis Sie schließlich ein Zeichen eingeben, mit dem dBase II zufrieden ist. Nachdem dBase II nun endlich eine Eingabe erhalten hat, mit der es arbeiten kann, prüft es nach, was es als Reaktion auf diese Eingabe tun soll. Dies geschieht mittels der DO-CASE-Verzweigung, die dBase II mitteilt, welche Aktionen von ihm bei welcher Antwort erwartet werden. Diese Verzweigung wird mit ENDCASE abgeschlossen.

CASE-Bedingung

Falls nun der Variablen »antwort« vom Benutzer das Zeichen »0« gegeben wurde, löscht dBase II den Bildschirm (ERASE), schreibt dem Benutzer die Mitteilung auf den Bildschirm, daß es die Adressendatei öffnet und führt dies sowohl bei der Adressendatei als auch der Indexdatei aus.

USE adressen INDEX name

Damit sind die Daten, wenn irgendwelche Manipulationen an der Datei vorgenommen werden, zugriffsbereit und liegen anschließend noch in alphabetisch geordneter Form vor.

PACK

löscht alle als gelöscht markierte Datensätze endgültig aus der Datei. Die Indexdatei wird dabei automatisch aktualisiert.

ERASE

löscht den Bildschirm und

QUIT

führt dazu, daß alle offenen Dateien geschlossen werden, dBase II sich verabschiedet und wir uns wieder im Betriebssystem CP/M befinden. Das erkennen wir daran, daß das A> wieder erscheint und auf Befehlseingaben wartet. Es wäre auch möglich, die PACK-Prozedur jeweils im Anschluß an den Programmpunkt »Datensätze löschen« durchzuführen, jedoch kann dies bei großen Datenbanken schon einige Zeit in Anspruch nehmen, wodurch man bei jedem Aufruf diese Zeit brauchen würde, während in der vorliegenden Form höchstens ein Mal gePACKt wird.

Sollte nun die Antwort nicht »0« sein, sondern »1«, werden von dBase II alle Befehle ausgeführt, die nach CASE antwort="1" folgen. Im vorliegenden Fall handelt es sich dabei um einen einzigen Befehl.

DO programmname

dBase II soll also, falls die Antwort des Benutzers »1« lautet, die im Programm »EINGEBEN.CMD« enthaltenen Befehle ausführen. Wir rufen also vom Programm »MENU.CMD« ein weiteres Programm auf, das seinerseits wieder ein Programm aufrufen kann, das seinerseits wieder ein ...

Die weiteren Zeilen geben dBase II vor, welche Befehle es ausführen soll, wenn die Antwort »2«, »3«, »4« oder »5« lautet, wodurch jedesmal ebenfalls ein weiteres Programm aufgerufen wird. Vergleichbar ist diese Methode mit der Unterprogrammtechnik in PASCAL, oder, weniger elegant, mit dem Befehl GOSUB in Basic. Diese Technik der Programm-Module ist sehr vorteilhaft, da die einzelnen Programme kürzer und damit überschaubarer sind als ein großes, möglicherweise verschachteltes Gesamtprogramm, dessen Erstellung aus Speicherplatzgründen auch mit dem eingebauten dBase-II-Texteditor nicht möglich wäre. Es ist auch viel leichter, sollten Sie später Ihr Programm einmal abändern wollen, ein kleines Modul zu ändern; außerdem sind bei dieser Methode Programmierfehler leichter zu lokalisieren.

ENDCASE

bezeichnet das Ende der »was-soll-ich-tun-falls-«-Abfrage.

Wenn der Benutzer zum Beispiel »2« als Antwort eingegeben hätte, würde dBase II die im Programm »LOESCHEN.CMD« enthaltenen Anweisungen durchführen, wäre dann irgendwann ins Menü zurückgekehrt und zwar an die Stelle, an der es das Menü vorher verlassen hatte (nächster Befehl). Wenn nun zufällig in der Variablen »antwort« eines der Zeichen von »0« bis »5« stände, könnte dBase II nicht in die DO-WHILE-Schleife springen, da die dafür nötige Bedingung nicht erfüllt wäre. Durch STORE " " TO antwort sorgen wir dafür, daß die Variable »antwort« einen Wert erhält, der der DO-WHILE-Bedingung entspricht, so daß wir uns wieder im Menü befinden.

Wenn Sie nun das erste Listing eingegeben haben, überprüfen Sie es noch einmal, da dies eine gute Möglichkeit ist, sich mit den Cursor-Steuerbefehlen vertraut zu machen.

Gehen Sie mit den Cursor-Tasten zum Anfang des Programmtextes, oder, falls dies bei Ihrer dBase-II-Version möglich ist, mittels des Befehls <CONTROL+R>. Vergleichen Sie die einzelnen Programmzeilen, indem Sie sich mittels <CONTROL+C> jeweils um einen halben Bildschirm in Richtung Programmende bewegen. Sollten Sie irgendwo mitten in einer Zeile etwas vergessen haben, steuern Sie den Cursor mit den grauen Steuertasten zu der Stelle, wo eingefügt werden soll, schalten Sie mit <CONTROL+V> in den Einfügemodus (wird von dBase II angezeigt) und beginnen zu schreiben. Sie werden feststellen, daß sich der Rest der Programmzeile nach rechts verschiebt, um Platz zu machen. Bevor Sie <RETURN> drücken, verlassen Sie den Einfügemodus durch Eingabe von <CONTROL+V>. Eine weitere Möglichkeit, den Einfügemodus zu verlassen, besteht darin, die aktuelle Bildschirmzeile mit Hilfe einer der grauen Cursor-Steuertasten nach oben oder unten zu verlassen. Drücken Sie auf keinen Fall die <RETURN>-Taste, während Sie sich in diesem Modus befinden, da dann automatisch ein zusätzliches Zeilenende-Zeichen eingefügt würde. Sie sind nun in der letzten Zeile und wollen Ihr Programm speichern. Dies gelingt Ihnen durch die Eingabe von <CONTROL+W> oder einfacher durch Drücken der kleinen Cursor-Taste direkt neben der rechten <SHIFT>-Taste. dBase II speichert nun Ihr Programm unter dem Namen MENU.CMD auf Diskette. Anschließend erscheint wieder ».«, das Zeichen für Sie, daß Sie sich im Direktmodus von dBase II befinden. Sollten Sie sich Ihr Programm noch einmal ansehen wollen, geben Sie

MODIFY COMMAND menu <RETURN>

oder einfacher

MODI COMM menu

ein.

Nach einer kurzen Ladezeit sehen Sie den Anfang Ihres Programms auf dem Bildschirm. Wenn Sie nun Veränderungen vornehmen und – wenn Sie damit fertig sind – <CONTROL+W> oder die Cursor-Taste neben der rechten <SHIFT>-Taste drücken, wird Ihr geändertes Programm unter dem Namen MENU.CMD gespeichert, während das ursprüngliche Programm unter dem Namen MENU.BAK als Sicherheitskopie auf der Diskette bleibt. Wenn Sie ganz sicher sind, daß Sie die Sicherheitskopie nicht mehr brauchen, können Sie diese durch DELETE FILE menu.bak <RETURN> löschen.

Alle Backup-Dateien löschen Sie durch Eingabe von

DELETE FILE *.bak <RETURN>

Seien Sie hierbei jedoch sehr vorsichtig, da Löschungen mit normalen Mitteln nicht mehr rückgängig zu machen sind! Im Laufe Ihrer Arbeit mit dBase II wird es vielleicht auch einmal vorkommen, daß Sie eine .CMD-Datei löschen und eine .BAK-Datei zur .CMD-Datei machen wollen. Gehen Sie dazu folgendermaßen vor:

DELETE FILE programmname .CMD <RETURN>

RENAME programmname.BAK TO programmname.CMD <RETURN>

Auch diese Befehlsfolge sollten Sie nur anwenden, wenn Sie sich ganz sicher sind. Eine kleine Geschichte soll an dieser Stelle zeigen, wie schnell eine kleine Unaufmerksamkeit die Arbeit von Wochen zunichte machen kann.

Der Autor diese Berichts hatte schon drei Wochen lang an einem dBase-II-Programm gearbeitet und wollte alle .BAK-Dateien, die sich in dieser Zeit angesammelt hatten, löschen, um neuen Platz auf der Diskette zu schaffen. Halb in Gedanken, der Kaffee wartete schon in der Küche, gab er DELETE FILE ????????.??? <RETURN> ein und löschte – die gesamte Diskette! Na ja, seitdem endet jede dBase-II-Sitzung mit dem »Ziehen« einer Sicherheitskopie, obwohl seitdem natürlich nichts mehr passiert ist.

Gehen Sie darum mit dem Löschbefehl von dBase II sehr sorgfältig um.

Um wieder in den Texteditor zu gelangen, geben Sie

MODI COMM eingeben

(Listing 2) ein. Nun können Sie die einzelnen Programmzeilen abtippen, wobei wir Ihnen jeweils wieder die einzelnen Befehle erklären werden.

STORE '...' TO ...

Zunächst belegen Sie alle im Verlauf dieses Moduls benötigten Variablen mit Werten, so daß dBase II weiß, wieviele Zeichen für die einzelnen Variablen benötigt werden. Für die Variablennamen wählen wir die gleichen Namen wie bei den Feldern, stellen jedoch ein »v:« voran, um sie von diesen zu unterscheiden. Die Variable »antwort« belegen wir mit »a«, damit im Menü, sollten Sie aus Versehen die <RETURN>-Taste berühren, automatisch das Menü wieder abgefragt wird. In der späteren Abfrage wird uns als Anrede »Herrn« angeboten, diese Variable können Sie natürlich je nach Art Ihres Bekanntenkreises auch mit »Frau , »Frl. « oder »Fam. « vorbesetzen, wobei Sie bei den drei letztgenannten die zusätzliche Leerstelle nicht vergessen sollten. Die Variablen »v:name«, »v:vorname«, »v:strasse« und »v:plzort« besetzen wir mit jeweils 20 Leerzeichen vor, da wir auch in unserer Datei dafür 20 Leerzeichen vorgesehen haben. »v:gebdat« wird mit 8, »v:telefon« mit 15, »v:notiz« mit 40 Leerzeichen vorbesetzt. Die folgende DO-WHILE-Schleife kennen wir schon. Solange die Bedingung der Schleife erfüllt ist, werden die Befehle, die zwischen DO WHILE und ENDDO stehen, ausgeführt. Wenn nun also die Variable »antwort« mit »a« oder »A« besetzt ist, was wir mittels des STORE-TO-Befehls erreicht haben, wird dBase II den Bildschirm löschen und uns einen leeren Datensatz zeigen, der durch die SAY-Befehle auf den Schirm gebracht wird. dBase II erwartet nun unsere Eingabe im ersten Feld des Datensatzes. Wenn dieses ausgefüllt ist, können Sie <RETURN> drücken und gelangen so zum nächsten Feld. Die grauen nebeneinanderliegenden Cursor-Tasten ermöglichen es Ihnen jedoch, jederzeit wieder zurück, nach rechts, links oder unten zu gehen. Dies ist jedoch nicht mehr möglich, wenn Sie das letzte Feld mit <RETURN> abgeschlossen haben, denn dies nimmt dBase II zum Anlaß, Ihre Eingaben zu lesen und Sie zu fragen, was es mit diesen Eingaben machen soll. Es bietet Ihnen »a« für Ändern als Möglichkeit an. Wenn Sie nun aus Versehen <RETURN> drücken, gibt es keinen Grund zur Aufregung, denn aufgrund der folgenden DO CASE/ENDCASE-Abfrage beendet dBase II die DO-WHILE-Schleife und wiederholt den eben durchgeführten Programmpunkt, wobei die von Ihnen eingegebenen Daten jedoch erhalten bleiben.

Anschließend werden Sie dann wieder nach Ihren Wünschen gefragt. Wählen Sie nun »m« oder »M«, trifft der erste Fall der DO CASE/ENDDO-Abfrage zu.

RELEASE ALL

dBase II löscht auf einen Schlag alle Variablen, die Sie ja nun nicht mehr brauchen. Leider wird dadurch auch die Variable »antwort« gelöscht, die Sie in jedem Programmpunkt benötigen, darum definieren wir diese wieder neu.

<RETURN>

Wir kehren wieder ins Hauptmenü zurück. Haben Sie statt »m« oder »M« jedoch »s« oder »S« für Speichern gewählt, sind einige Befehle mehr notwendig, daß dBase II die von Ihnen gewünschte Arbeit verrichtet.

USE adressen INDEX name

Die Datei »adressen« wird eröffnet, da sie sonst nicht bearbeitet werden kann. Gleichzeitig wird die Indexdatei »name« angesprochen, damit auch neu hinzugefügte Datensätze gleich ihren korrekten Platz in der Datei einnehmen.

APPEND BLANK

Der Datei wird ein leerer (BLANK) Datensatz angehängt (APPEND).

REPLACE anrede WITH v:anrede

Nun ersetzt (REPLACE) dBase II das leere Feld »anrede«

durch die Variable »v:anrede«. So werden nacheinander die leeren Felder des neuen Datensatzes durch Ihre Eingaben ersetzt und – nachdem das letzte Feld entsprechend behandelt ist – die Datei mit USE geschlossen und der Bildschirm gelöscht.

Es folgt nun ein kleines Untermenü, das den einzigen Zweck hat, Sie zu fragen, ob Sie weitere Datensätze eingeben wollen. Da Sie eine Zeile vorher die Variable »antwort« mit »N« vorbesetzt haben, brauchen Sie nur <RETURN> zu drücken und gelangen wieder ins Hauptmenü, da dann die Bedingung in der folgenden

IF

-Abfrage zutrifft, daß Ihre Antwort ungleich »j« und »J« ist. Daher werden alle Variablen gelöscht, die Variable »antwort«, die wir im Hauptmenü brauchen, mit » « besetzt und dBase II führt uns ins Hauptmenü zurück.

ENDIF

zeigt das Ende der Reihe von Befehlen an, die von dBase II ausgeführt werden, falls die hinter IF stehenden Bedingungen zutreffen.

Sollten Sie die Frage, ob Sie weitere Datensätze eingeben wollen, mit »j« oder »J« beantworten, wird die IF/ENDIF Schleife übersprungen, die Fallabfrage DO CASE beendet (ENDCASE), die Variable »antwort« erhält den Wert »a«, das Ende der DO-WHILE-Schleife wird erreicht (ENDDO), die Bedingung für das Abarbeiten dieser Schleife trifft damit zu und die Bildschirmmaske für das Eingeben neuer Datensätze erscheint erneut.

Es könnte Sie möglicherweise stören, daß in der Maske die Daten erscheinen, die Sie als letzte eingegeben haben, jedoch erleichtert dies Ihnen die Entscheidung, ob Sie einen bestimmten Datensatz schon eingegeben haben oder nicht. Um die Ausgabe der letzten Daten zu unterdrücken, müßten Sie vor der zweitletzten Zeile des Programms noch einmal die STORE-TO-Befehle 2 bis 9 vom Programmanfang eingeben.

Das Unterprogramm »loeschen«

Inzwischen wissen Sie ja, wie man in den Texteditor von dBase II gelangt. Ihr neues Programm-Modul heißt loeschen (Listing 3). Im folgenden werden jetzt nur noch die Befehle, die Sie noch nicht kennengelernt haben, in allen Einzelheiten besprochen.

Besetzen Sie die Variable »antwort« mit »j«, die Variable »v:name« mit 20 Leerzeichen. Sie brauchen »v:name« später als Suchkriterium für einen Datensatz. Öffnen Sie nun die Datei »adressen« mit der Indexdatei »name«, damit das Suchen erleichtert wird und eventuelle Löschungen gleich auch in der Indexdatei registriert werden. Die folgende DO WHILE/ENDDO-Schleife kennen Sie ja bereits.

GO TOP

setzt den Zeiger von dBase II an den Anfang der Datei. Dieser Befehl ist wichtig, da sonst dBase II bei erneutem Suchbefehl eventuell an der Stelle, an der es zuletzt war, weitersuchen würde. Nun wird der Bildschirm gelöscht, dBase II möchte von Ihnen den gesuchten Familiennamen erfahren (GET) und liest diesen Namen (READ).

LOCATE FOR name=v:name

Dieser Befehl gibt die Anweisung vom Anfang der Datei an, in der Reihenfolge, die in der Indexdatei festgelegt ist, den Datensatz zu suchen, dessen Feld »name« identisch ist mit der Variablen »v:name«. Sollten Sie den Namen falsch geschrieben haben, wird dBase II den entsprechenden Datensatz natürlich nicht finden. Vielleicht machen Sie sich einmal die Mühe (oder die Freude), wenn Sie vertrauter sind mit dBase II und seiner großen Befehlsvielfalt, eine Suchroutine mit Jokerfunktion unter dBase II selbst zu erstellen.

Sollte dBase II bei seiner Suche erfolglos sein, würde das Ende der Datei erreicht,

EOF

also wahr und die gesamte DO WHILE/ENDDO-Schleife übersprungen werden. Dazu jedoch später. Sollte jedoch das Dateiende nicht erreicht sein (.NOT. EOF) und das Feld »name« entspricht dem eingegebenen Namen, der in die Variable »v:name« eingegeben wurde, listet dBase II alle Felder des Datensatzes an der von Ihnen gewünschten Stelle des Bildschirms auf.

In Zeile 5, Spalte 5 lernen wir einen neuen Formatierungsbefehl für Daten kennen:

TRIM (name)

Manchmal wird es nötig sein, zwei oder mehr Felder eines Datensatzes zusammen auszudrucken, wie zum Beispiel Vorname und Familienname. Erinnern wir uns daran, daß unsere Felder »name« und »vorname« jeweils 20 Zeichen lang sind. Sind die Einträge in diesen Feldern kürzer, so wird der Rest mit Leerstellen aufgefüllt, was bei dem Befehl

... SAY vorname+name

zu folgendem unschönen Bild führen würde:

Paul   Meier

Die TRIM-Funktion bietet nun die Möglichkeit, unnötige Leerstellen am Ende des Feldes abzuschneiden. Doch ist sie auch nicht ohne Tücken, denn wenn nun der Befehl in abgeänderter Form lautet

... SAY TRIM(vorname)+name

erhalten wir als Ergebnis:

PaulMeier

Das ist zwar ärgerlich, doch woher soll dBase II wissen, daß wir uns hier eine Leerstelle zwischen den beiden Feldern wünschen. Also erweitern wir unseren Befehl nochmals um einen kleinen Zusatz, wodurch unser Paul Meier uns dann wunderschön formatiert vom Bildschirm anblickt:

... SAY TRIM(vorname)+" "+name

Vielleicht werden Sie sich fragen, wieso nun plötzlich alle Daten auf dem Bildschirm auftauchen, obwohl wir am Anfang des Programms keine entsprechenden Variablen vorbesetzt oder definiert haben. Dies ist jedoch hier nicht nötig, da die Feldnamen direkt von dBase II ausgegeben werden, und diese Feldnamen haben mit unseren ähnlich lautenden Variablennamen überhaupt nichts zu tun.

Sehen Sie sich nun den Datensatz, den dBase II für Sie gefunden hat, in aller Ruhe an. Dann entscheiden Sie sich, ob dies der Datensatz ist, den Sie löschen wollen – dann geben Sie »l« oder »L« ein –, ob dBase II weitersuchen soll – befehlen Sie dies mit »w« oder »W« –, oder ob Sie ins Hauptmenü zurückwollen – Eingabe von »m« oder »M«. Wenn Sie durch Ihre Eingabe zeigen, daß Sie ins Hauptmenü zurückwollen, schließt dBase II im Rahmen der folgenden DO CASE/ENDCASE-Abfrage die Datei, löscht die Variable »v:name« aus dem Speicher, belegt »antwort« mit » « und führt zurück ins Hauptmenü. Wenn Sie den Datensatz löschen wollen (»l« oder »L«), kommt der Befehl

DELETE

zur Ausführung, der den auf dem Bildschirm angezeigten Datensatz mit einer Löschmarkierung versieht. Zwar ist dieser Datensatz noch auf der Diskette vorhanden, jedoch für Sie nicht mehr sichtbar, da Sie dies im Hauptmenü mit dem Befehl SET DELETED ON angeordnet haben. Wenn Sie die Sitzung mit dBase II beenden, werden alle mit Löschmarkierung versehenen Datensätze endgültig gelöscht.

Sollte dBase II nach dieser Löschung noch nicht das Dateiende erreicht haben, sucht es nach weiteren Namen, die Ihrer Eingabe entsprechen. Bei jedem dieser Namen wird erneut gefragt, ob der Datensatz gelöscht werden soll oder nicht. So können Sie ganze Familien, mit denen Sie verfeindet sind, aus Ihrem »Gedächtnis« streichen, andererseits wird aber durch die immer wiederholte Frage, ob der Datensatz tatsächlich gelöscht werden soll, verhindert, daß unabsichtlich noch gebrauchte Datensätze auf Nimmerwiedersehen verschwinden.

CONTINUE
bedeutet für dBase II die Anweisung, nach einem LOCATE-FOR-Befehl nach weiteren Datensätzen zu suchen, die der formulierten Bedingung entsprechen.

Wenn dann schließlich – falls Sie dies wollen – die ganze Datei durchsucht und das Dateiende (EOF) erreicht ist, werden Sie gefragt, ob Sie weitere Datensätze löschen wollen. Die Bejahung dieser Frage führt wieder zur Bildschirmmaske des Moduls LOESCHEN; wird die Frage verneint, kehren wir zum Hauptmenü zurück.

Speichern Sie nun Ihr Programm-Modul ab.

Das Unterprogramm »aendern«

Die Eingabe dieses Programm-Moduls (Listing 4) beginnen Sie in der Ihnen nun schon geläufigen Form

MODIFY COMMAND aendern

Wie bei allen der vorher eingegebenen Module beginnen Sie damit, daß Sie den im Programmverlauf benötigten Variablen Werte zuweisen. »antwort« erhält diesmal den Wert »j«, da wir diesen Wert sehr wahrscheinlich später brauchen werden. Die zweite benötigte Variable ist wieder der Familienname »v:name«, da dieser unser Suchkriterium darstellt. Diese Variable besetzen wir wieder mit 20 Leerzeichen, da, wie Sie sich erinnern werden, das Feld Name in unserer Datei ebenfalls 20 Leerzeichen besetzt. In der bekannten Form öffnen wir nun die Datei »adressen« unter gleichzeitiger Öffnung der Indexdatei »name«. Da die Bedingung für den Eintritt in die DO WHILE/ENDDO-Schleife durch den Wert »j« für »antwort« erfüllt ist, geben wir zunächst den Befehl, den Zeiger auf den Anfang der Datei zu setzen. Zwar wird dies schon durch das Öffnen der Datei bewirkt, jedoch muß gewährleistet sein, daß bei wiederholtem Durchlaufen der Schleife die Datei jedes Mal von Anfang an durchsucht wird. Nach dem Löschen des Bildschirms wird nach dem Familiennamen gefragt, diese Zeichenfolge wird gelesen und der Variablen »v:name« zugeordnet. Dann sucht dBase II den ersten Datensatz, dessen Feld »name« mit der Variablen »v:name« identisch ist. Findet es keinen solchen Datensatz, wird die Schleife (ENDDO) beendet, und Sie entscheiden sich fürs Weitersuchen oder Aufhören. Findet dBase II einen passenden Datensatz, gibt es alle Felder des Datensatzes auf dem Bildschirm aus. Sie entscheiden sich nun, ob Sie die Suche abbrechen – »m« oder »M« –, weitersuchen – »w« oder »W«, oder den Datensatz oder Teile des Datensatzes ändern wollen.

Wenn Sie sich im Rahmen der folgenden DO CASE/END-CASE-Abfrage für den Abbruch der Suche entscheiden, werden die Datei und die Indexdatei mit USE geschlossen, die Variablen gelöscht, die Variable »antwort« mit » « vorbesetzt und dBase II kehrt ins Hauptmenü zurück.

Entscheiden Sie sich fürs Ändern, müssen wir zunächst die Inhalte der einzelnen Felder des gefundenen Datensatzes auf Variable übertragen, die wir dann zunächst ohne bleibende Folgen bearbeiten können. Da wir uns fürs Ändern entschieden haben, besitzt unsere Variable »antwort« im Augenblick den Wert »a«. Damit ist die Bedingung für den Eintritt in die zweite DO WHILE/ENDDO-Schleife erfüllt, in der dBase II zunächst den Bildschirm löscht und dann unsere neu angelegten Variablen zeigt, die wir beliebig ändern, löschen oder unverändert lassen können. Auch hier können Sie wieder mit Hilfe der grauen Cursor-Tasten beliebig innerhalb der Felder herumwandern, bis Sie das letzte Feld mit <RETURN> abgeschlossen haben. Sollten Sie die anschließende Frage, ob alles richtig ist, mit einer anderen Eingabe als »j« oder »J« beantworten, wird die letzte Schleife wiederholt, jedoch schlägt Ihnen dBase II für die einzelnen Felder schon Ihre letzten Eingaben vor. Wenn Ihre Eingaben korrekt sind, wird die zweite DO-WHILE-Schleife verlassen, und dBase II ersetzt die Felder des gefundenen Datensatzes mit den entsprechenden geänderten oder unveränderten Variablen und schlägt gleichzeitig vor, weiterzusuchen.

Sollten Sie »w« oder »W« für Weitersuchen eingegeben haben, wird die eingeschlossene DO-WHILE-Schleife gar nicht erst bearbeitet, sondern die Suche mit CONTINUE geht weiter, bis ein neuer Datensatz gefunden wird, auf den die Bedingung »name=v:name« zutrifft.

Wenn Sie sich nun entschließen, keine weiteren Änderungen durchzuführen, werden alle Variablen gelöscht (RELEASE ALL), die Datei und die Indexdatei geschlossen, die Variable antwort mit » « vorbesetzt und dBase II kehrt zum Hauptmenü zurück.

Speichern Sie nun wieder Ihr Programm.

Das Programm »ausgabe«

»MODIFY COMMAND ausgabe« (Listing 5) lautet hier wieder der übliche Befehl, um in den Texteditor von dBase II zu gelangen.

Wie immer benötigen wir »antwort« als Variable, diesmal jedoch zusätzlich die Variable »anfang«, in der wir später festlegen, ab welchem Buchstaben Datensätze auf dem Bildschirm gezeigt werden sollen.

Nun öffnen wir wieder unsere Datei »adressen« und die Indexdatei »name« und beginnen mit der DO-WHILE-Schleife, in der Sie zunächst nach dem Buchstaben, ab dem geblättert werden soll, gefragt werden. Hier wurde »0« als Abbruchbedingung und für die Rückkehr ins Menü gewählt, da »m« oder »M« Suchbuchstabe sein kann.

Bei Eingabe von »0« wird die Variable »anfang« gelöscht, »antwort« mit » « besetzt, werden die Dateien geschlossen und dBase II kehrt ins Hauptmenü zurück.

Die Eingabe eines anderen Buchstabens führt zum Überspringen der IF/ENDIF-Bedingung und damit zur Ausführung des Befehls

STORE !(anfang) TO anfang

Damit wird ein Buchstabe unabhängig von seiner Schreibweise (groß/klein), in einen Großbuchstaben verwandelt. Würde dies nicht geschehen, könnte bei Eingabe eines Kleinbuchstabens dBase II niemals den entsprechenden Datensatz finden, da natürlich alle Namen mit einem Großbuchstaben beginnen.

Nachdem GO TOP den Zeiger auf den Anfang der Datei gesetzt hat, führt dBase II den nächsten Befehl aus.

LOCATE FOR (name,Beginn,Länge) = anfang

Das Dollarzeichen entspricht dem Basicbefehl MID, mit dem aus einem String bestimmte Zeichen herausgeschnitten werden. Aus dem Feld »name« wird hier ein Teil herausgeschnitten, der mit dem ersten Zeichen des Feldes beginnt und eine Länge von 1 besitzt, mit anderen Worten, der erste Buchstabe des Namens. Wenn nun der erste Buchstabe des Feldes »name« mit der Variablen »anfang« identisch ist, wird der gefundene Datensatz gezeigt. Sollte kein Datensatz mit dem Suchbuchstaben als Anfangsbuchstabe des Familiennamens in der Datei sein, wird der erste Datensatz mit dem nächsten Buchstaben gezeigt. Also: Falls kein Familienname mit »F« in der Datei ist, wird der erste Datensatz mit »G« ausgegeben. Nun können Sie sich entscheiden, ob Sie das Programm-Modul verlassen wollen, oder ob Sie in der Datei alphabetisch vorwärts oder rückwärts blättern wollen.

Entscheiden Sie sich durch Eingabe von »v« oder »V« für Vorwärtsblättern, kommt, falls das Ende der Datei noch nicht erreicht ist (IF .NOT. EOF), der Befehl

SKIP

zur Anwendung, der den Zeiger in der Datei um einen Datensatz weiter in alphabetischer Reihenfolge in Richtung auf das »Z« rückt. Ersatzweise könnten Sie hier auch den Befehl CONTINUE verwenden.

Nun wird das Ende der DO-WHILE-Schleife erreicht, da alle anderen Befehle übersprungen werden und der nächste Datensatz wird ausgegeben. Sollte der Zeiger von dBase II jedoch schon auf dem Dateiende stehen (IF EOF), trifft die nächste IF-Bedingung zu: dBase II teilt Ihnen mit, daß das

Dateiende erreicht ist und wartet auf einen beliebigen Tastendruck, um weiterzumachen.

Haben Sie sich fürs Rückwärtsblättern entschieden, kommt eine Variante des SKIP-Befehls zur Anwendung,

SKIP-1

der den Zeiger in der Datei um einen Datensatz zurück in umgekehrter alphabetischer Reihenfolge in Richtung auf das »A« rückt.

Wenn Sie dieses Programm-Modul verlassen wollen (0), wird die Variable »anfang« gelöscht, »antwort« wie üblich mit » « besetzt und Sie kehren ins Hauptmenü zurück.

Und jetzt speichern ... - Sie wissen ja schon.

Das Programm »drucken«

Das folgende Unterprogramm oder Programm-Modul (Listing 6) dient zum Ausdruck von Adreßlisten und Etiketten, wobei Sie frei bestimmen können, ab welchem Anfangsbuchstaben des Familiennamens und bis zu welchem Anfangsbuchstaben gedruckt werden soll. Durch die Definition der Variablen »anfang« und »ende« am Anfang des Programmteils schlägt Ihnen dBase II später »A« als Anfangsbuchstaben des ersten zu druckenden Familiennamens und »Z« als Anfangsbuchstaben des letzten zu druckenden Familiennamens vor, wodurch alle Datensätze von »A« bis »Z« gedruckt würden.

Da »antwort« im Augenblick den Wert » « besitzt, wird die DO-WHILE-Schleife begonnen und Sie können sich entscheiden, ob Sie Etiketten oder eine Adreßliste drucken oder ob Sie ins Hauptmenü zurückwollen.

Wenn Sie sich fürs Drucken entscheiden, bestimmen Sie, ab welchem und bis zu welchem Buchstaben gedruckt werden soll. Die beiden eingegebenen Buchstaben werden durch »!« in Großbuchstaben verwandelt, da auch die Familiennamen mit Großbuchstaben beginnen.

Sollten Sie sich für einen Abbruch entschieden haben, werden alle Variablen gelöscht, »antwort« mit » « besetzt und unser Hauptmenü erscheint wieder.

Bleiben wir jedoch beim Drucken und nehmen wir an, Sie hätten sich für den Druck einer Adressenliste entschieden, das heißt, daß Sie die entsprechende Frage mit »l« oder »L« beantwortet hätten. In der üblichen Weise eröffnen Sie die Datei »adressen« mit der Indexdatei »name«. LOCATE FOR sucht nach dem ersten Familiennamen, der mit dem Buchstaben, der in der Variablen »anfang« definiert wurde, beginnt oder, falls es einen solchen Namen nicht gibt, nach einem Namen, der mit dem in alphabetischer Reihenfolge nächsten Buchstaben beginnt.

SET PRINT ON

schaltet für die folgenden Ausgaben zusätzlich den Drucker ein, so daß Sie die Ausgaben gleichzeitig auf Bildschirm und Drucker verfolgen können, was Ihnen unnötige Verrenkungen zur Beobachtung des Druckers erspart.

? " " schaltet zunächst, da es ja dreimal nacheinander eingegeben wird, drei Leerzeilen. Hier genügt allerdings auch die Eingabe von »?« ohne anschließende Anführungszeichen.

Nun brauchen wir für unseren Ausdruck pro Seite die Überschrift, die uns zeigt, was eigentlich ausgedruckt wird. Die Formulierung können Sie natürlich frei wählen. Es empfiehlt sich, in dieser Überschrift mit

DATE()

das aktuelle Datum in die Überschrift einzufügen. Ihr Computer setzt hier das Datum ein, das Sie am Anfang des dBase-II-Laufs eingeben. Sollten Sie dBase II und Ihr Programm über die Datei PROFILE.SUB direkt von CP/M aus starten, müßten Sie allerdings vor der DO-WHILE-Schleife in Listing 1 folgende Zeilen einfügen:

```
store " " to datum
erase
@ 2, 5 say "Datum tt/mm/jj " get datum picture "99/99/99"
set date() to datum
release datum
```

GET PICTURE "99/99/99"

akzeptiert bei der Eingabe des Datums nur Zahlen (9). Diese sechs Zahlen werden hintereinander eingegeben, dBase II setzt automatisch die Schrägstriche.

SET DATE() TO DATUM

teilt dem System mit, daß als aktuelles Datum der Inhalt der Variablen »datum« gelten soll. Danach wird die Variable »datum« nicht mehr gebraucht, da auch nach einem RELEASE ALL jederzeit das aktuelle Datum »date()« abgefragt werden kann.

Doch zurück zu unserem Druckvorgang. Nun nimmt die Variable »nummer« den Wert 1 an und die DO-WHILE-Schleife beginnt. Wenn nun das Ende der Datei noch nicht erreicht ist und auch die Variable »nummer« einen Wert besitzt, der kleiner als 20 ist und gleichzeitig der Anfangsbuchstabe des Familiennamens des Datensatzes, der durch »anfang« und »ende« definierten Bedingung entspricht, werden 5 Leerzeichen gedruckt, darauf folgt der Inhalt des Feldes »anrede«, gekürzt auf 4 Zeichen, so daß statt »Herrn« hier nur »Herr« erscheint (beziehungsweise »Frau« oder »Frl.«), gefolgt von einem Leerzeichen. Darauf folgt der Vorname, bei dem die hinteren überflüssigen Leerzeichen durch TRIM abgeschnitten wurden, gefolgt von einem weiteren Leerzeichen. Der Befehl

??

zeigt an, daß in der gleichen Zeile weitergedruckt werden soll und zwar der Name und das Geburtsdatum. In der nächsten Zeile werden entsprechend der vorherigen Zeile die Felder »strasse«, »plzort« und »telefon« ausgedruckt. Die dritte Zeile des Datensatzes druckt entsprechende Bemerkungen. Jetzt werden Sie auch verstehen, warum nach 20 Datensätzen eine neue Seite begonnen wird: Der Vorspann nimmt 6 Zeilen in Anspruch, 20 Datensätze mit jeweils drei Zeilen ergeben 60 Zeilen, insgesamt also 66 Zeilen. Nachdem nun ein Datensatz gedruckt wurde, wird die Variable »nummer« um 1 auf 2 erhöht. Es werden nun solange Datensätze gedruckt, bis »nummer« größer als 20 geworden ist oder das Dateiende noch vor der 20 erreicht wurde, oder der Anfangsbuchstabe des Familiennamens alphabetisch nach dem für die Variable »ende« definierten Buchstaben liegt. Sollten die beiden letztgenannten Bedingungen nicht zutreffen, erfolgt durch

EJECT

ein Seitenvorschub. Nun wird wieder der Kopf mit dem aktuellen Datum gedruckt, die Prozedur wird von vorne wiederholt. Die Zahl der Datensätze ist relativ leicht zu überblicken, da jeweils genau 20 Datensätze auf eine Druckerseite gehen. Wenn alle gewünschten Daten ausgedruckt sind, erfolgt wieder ein Seitenvorschub zum Papieranfang. Die Druckerausgabe wird durch

SET PRINT OFF

beendet, so daß Ausgaben nur noch auf dem Bildschirm erfolgen. Sollten Sie »e« oder »E« für Etikettendruck eingegeben haben, so ist der Such- und Druckvorgang im Prinzip der gleiche, Sie müssen jedoch vorher eine kleine Rechnung durchführen. Wie Sie im Listing 6 sehen können, werden unter der Option Etiketten genau 9 Zeilen gedruckt. Dies ist darauf zurückzuführen, daß das in Deutschland verbreitete Endlospapier bei normalem Zeilenabstand mit 72 Zeilen bedruckt werden kann. Zählen Sie die Etiketten, die auf einem Blatt Druckpapier untereinander von Perforation zu Perforation Platz haben. Teilen Sie 72 durch diese Zahl und Sie erhalten die Zahl von Zeilen, die Ihnen zum Drucken auf einem Etikett zur Verfügung stehen. Sollten zum Beispiel auf einem Normblatt 12 Ihrer Etiketten Platz finden, so stehen Ihnen pro Etikett sechs Zeilen zur Verfügung, die Sie sinnvollerweise so aufteilen:

Leerzeile

Anrede
Vorname Name
Strasse Hausnummer
Postleitzahl Ort
Leerzeile

Als kleine Hilfe für unsere gestreßten Briefträger wird beim Etikettendruck durch den Befehl

!(name)

der Familienname des Adressaten in Großbuchstaben ausgedruckt. Beim Etikettendruck ist es unnötig, die Datensätze zu zählen, da hier keine Rücksicht auf einen oberen oder unteren Rand genommen werden muß und außerdem die Seitenüberschrift entfällt. Nachdem Sie nun auch noch die letzten Zeilen unseres sechsten Listings eingegeben und das Modul abgespeichert haben, ist es endlich soweit! Starten Sie das Programm mit

DO MENU

und folgen Sie ganz einfach den Anweisungen. Übrigens, für die Statistiker unter Ihnen: unser kleines Adressenprogramm ist in der vorliegenden Form in der Lage, mit einer Floppy 1571 ungefähr 1800, 1570 ungefähr 600 Datensätze zu verwalten.

Wenn Sie alle Möglichkeiten des Programms durchgespielt und wie ich hoffe, keinen Fehler entdeckt haben, können Sie, falls außer Ihnen auch noch andere das Programm benutzen, die <ESCAPE>-Taste sperren. Fügen Sie daher im Listing 1 als erste Programmzeile ein

SET ESCAPE OFF

Dadurch wird es dem Benutzer unmöglich gemacht, die Ausführung einer Befehlsdatei mittels <ESCAPE>-Taste zu unterbrechen. Zum Abschluß dieser kleinen Entdeckungsreise in dBase II als Programmiersprache noch ein paar Vorschläge, wie Sie unser kleines Adressenprogramm erweitern könnten:

- bauen Sie mehrere Indexfelder ein
- drucken Sie Listen in Abhängigkeit von diesen Feldern
- schützen Sie die Benutzung der Dateien durch ein Paßwort
- programmieren Sie die Jokerfunktion fürs Suchen
- programmieren Sie eine Textverarbeitung, die Serienbriefe verschickt
- schreiben Sie eine Autostartroutine mit PROFILE.SUB.

Der Möglichkeiten gibt es viele, jedoch würde es zu weit führen, sie alle hier aufzuzählen.

Also, sprechen Sie dBase II?

(Henk Driessen/bj)

```
* -----------------------------------------------
* Listing Nr. 1
* ----
* MENU
* ----
* dient als Menue fuer die Adressenverwaltung
* -----------------------------------------------
*
set deleted on
set talk off
store " " to antwort

do while antwort<"0" .or. antwort >"5"

  erase
  @  2, 5 say "M E N U E"
  @  3, 5 say "-------------------------------"
  @  5, 5 say "0. Programm beenden"
  @  7, 5 say "1. neue Datensaetze eingeben"
  @  9, 5 say "2. Datensaetze loeschen"
  @ 11, 5 say "3. Datensaetze aendern"
  @ 13, 5 say "4. Datensaetze auf Bildschirm"
  @ 15, 5 say "5. Adressenliste drucken"
  @ 17, 5 say "-------------------------------"
  @ 19, 5 say "Druecken Sie die entsprechende"
  @ 20, 5 say "Zahlentaste zur Auswahl!     " get antwort

  read

    do case

      case antwort="0"
        erase
        @  2, 5 say "Adressendatei wird eroeffnet."
        use adressen index name
        @  4, 5 say "Geloeschte Datensaetze werden entfernt."
        @  6, 5 say "Adressendatei wird neu indiziert"
        pack
        erase
        quit

      case antwort="1"
        do eingeben

      case antwort="2"
        do loeschen

      case antwort="3"
        do aendern

      case antwort="4"
        do ausgabe

      case antwort="5"
        do drucken

    endcase

  store " " to antwort
  enddo
```

Listing 1. »menu« dient als Abfragemaske für die Adreßverwaltung. Bitte verwenden Sie zur Eingabe aller Listings den Texteditor von dBase II (siehe Text).

```
* ==================================
* Listing Nr. 2
* --------
* EINGEBEN
* --------
* dient der Eingabe neuer Datensaetze
* ----------------------------------
*
store "a"                        to antwort
store "Herrn"                    to v:anrede
store "                        " to v:name
store "                        " to v:vorname
store "                        " to v:plzort
store "                        " to v:strasse
store "        "                 to v:gebdat
store "              "           to v:telefon
store "                                           " to v:notiz

do while antwort="a" .or. antwort="A"

  erase

  @  2, 5 say "Anrede        " get v:anrede
  @  3, 5 say "Vorname       " get v:vorname
  @  4, 5 say "Name          " get v:name
  @  6, 5 say "Geburtsdatum  " get v:gebdat picture "99.99.99"
  @  8, 5 say "Strasse/Hnr.  " get v:strasse
  @  9, 5 say "PLZ/Wohnort   " get v:plzort
  @ 10, 5 say "Telefon       " get v:telefon
  @ 12, 5 say "Bemerkungen   " get v:notiz
  read

  @ 15, 5 say "Ueberpruefen Sie Ihre Eingaben!"
  @ 17, 5 say "'s' fuer Speichern"
  @ 18, 5 say "'a' fuer Aendern"
  @ 19, 5 say "'m' fuer Menue      " get antwort
  read

  do case

    case antwort="m" .or. antwort="M"
      release all
      store " " to antwort
      return

    case antwort= "a" .or. antwort="A"
      enddo

    case antwort="s" .or. antwort="S"
      use adressen index name
      append blank
      replace anrede  with v:anrede
      replace name    with v:name
      replace vorname with v:vorname
      replace strasse with v:strasse
      replace plzort  with v:plzort
      replace telefon with v:telefon
      replace gebdat  with v:gebdat
      replace notiz   with v:notiz
      use

      erase
      store "n" to antwort
      @  2, 5 say "Weitere neue Datensaetze eingeben? j/n " get antwort
      read
```

```
      if antwort <>"j" .and. antwort<>"J"
        release all
        store " " to antwort
        return
      endif

    endcase

store "a" to antwort

enddo

return
```

Listing 2. Dieses Unterprogramm dient der Eingabe neuer Datensätze

```
* ----------------------------------
* Listing Nr. 3
* --------
* LOESCHEN
* --------
* dient zum Loeschen von Datensaetzen
* ----------------------------------
*
store "j" to antwort
store "                    " to v:name
use adressen index name

do while antwort="j" .or. antwort="J"
  go top
  erase
  @  2, 5 say "Familienname " get v:name
  read

  locate for name=v:name

 do while .not. eof

  if name=v:name
    @  5, 5 say trim(name)+", "+trim(vorname)
    @  6, 5 say gebdat
    @  8, 5 say strasse
    @  9, 5 say plzort
    @ 10, 5 say telefon
    @ 12, 5 say notiz
    @ 15, 5 say "Geben Sie ein:"
    @ 17, 5 say "'l' fuer Loeschen"
    @ 18, 5 say "'w' fuer Weitersuchen"
    @ 19, 5 say "'m' fuer Menue"
    @ 21, 5 say "        " get antwort
    read

    do case

      case antwort="m" .or. antwort="M"
       use
       release v:name
       store " " to antwort
       return

       case antwort="l" .or. antwort="L"
       delete
       store "w" to antwort
       continue

      case antwort="w" .or. antwort="W"
       continue

    endcase
  endif

 enddo

 store " " to antwort
 erase

 @  2, 5 say "Weitere Loeschungen j/n " get antwort
 read

 if antwort<>"j" .and. antwort<>"J"
   release v:name

      use
      store " " to antwort
      return
    endif

   enddo
   return
```

Listing 3. »loeschen« entfernt Datensätze aus der Datenbank

```
* ----------------------------------
* Listing Nr. 4
* -------
* AENDERN
* -------
* dient zum Aendern von Datensaetzen
* ----------------------------------
*
store "j" to antwort
store "                    " to v:name

use adressen index name

do while antwort="j" .or. antwort="J"
  go top
  erase
  @  2, 5 say "Familienname " get v:name
  read

  locate for name=v:name

 do while .not. eof

  if name=v:name
    @  3, 5 say "Vorname      : " + vorname
    @  4, 5 say "Geburtsdatum : " + gebdat
    @  6, 5 say "Strasse/Hnr. : " + strasse
    @  8, 5 say "PLZ/Ort      : " + plzort
    @  9, 5 say "Telefon      : " + telefon
    @ 11, 5 say "Bemerkungen  : " + notiz

    @ 15, 5 say "Geben Sie ein:"
    @ 17, 5 say "'a' fuer Aendern"
    @ 18, 5 say "'w' fuer Weitersuchen"
    @ 19, 5 say "'m' fuer Menue"
    @ 21, 5 say "        " get antwort
    read

    do case

      case antwort="m" .or. antwort="M"
       use
       release all
       store " " to antwort
       return
     case antwort="a" .or. antwort="A"
       store anrede   to v:anrede
       store name     to v:name
       store vorname  to v:vorname
       store gebdat   to v:gebdat
       store strasse  to v:strasse
       store plzort   to v:plzort
       store telefon  to v:telefon
       store notiz    to v:notiz

       do while antwort<>"j" .and. antwort<>"J"
         erase
         @  2, 5 say "Anrede        " get v:anrede
         @  3, 5 say "Vorname       " get v:vorname
         @  4, 5 say "Name          " get v:name
         @  6, 5 say "Geburtsdatum  " get v:gebdat
         @  8, 5 say "Strasse/Hnr.  " get v:strasse
         @  9, 5 say "PLZ/Ort       " get v:plzort
         @ 10, 5 say "Telefon       " get v:telefon
         @ 12, 5 say "Bemerkungen   " get v:notiz
         read
         @ 15, 5 say "Alles richtig?  j/n " get antwort
         read
       enddo

       replace anrede   with v:anrede
       replace name     with v:name
       replace vorname  with v:vorname
       replace gebdat   with v:gebdat
       replace strasse  with v:strasse
```

Listing 4. »aendern« ist eines der fünf menügesteuerten Unterprogramme

```
        replace plzort   with v:plzort
        replace telefon  with v:telefon
        replace notiz    with v:notiz

        store "w" to antwort
        continue

      case antwort="w" .or. antwort="W"
        continue

    endcase
  endif

enddo

store " " to antwort
erase
@  2, 5 say "Weitere Aenderungen j/n " get antwort
read

if antwort<>"j" .and. antwort<>"J"
  release all
  use
  store " " to antwort
  return
endif

enddo
return
```

Listing 4. (Schluß). Mit diesem Modul können fehlerhafte Datensätze korrigiert werden

```
* -------------------------------------------------------
* Listing Nr. 5
* -------
* AUSGABE
* -------
* dient zur Ausgabe von Datensaetzen auf dem Bildschirm
* -------------------------------------------------------
*
store " " to anfang
store " " to antwort
use adressen index name

do while anfang <> "0"
    erase
    @  2, 5 say "Geben Sie den Buchstaben ein, ab dem in der Datei"
    @  3, 5 say "geblaettert werden soll."
    @  4, 5 say "'0' fuehrt ins Menue zurueck."
    @  7, 5 say "Anfangsbuchstabe " get anfang
    read

  if anfang = "0"
    release anfang
    store " " to antwort
    use
    return
  endif

    store !(anfang) to anfang

    go top
    locate for $(name,1,1) >= anfang
    erase

  do while antwort <> "0"
      erase
      @  2, 5 say "Anrede        : " + anrede
      @  3, 5 say "Name          : " + trim(vorname) + " " + trim(name)
      @  5, 5 say "Geburtsdatum : " + gebdat
      @  7, 5 say "Strasse/Hnr. : " + strasse
      @  8, 5 say "PLZ/Wohnort  : " + plzort
      @  9, 5 say "Telefon      : " + telefon
      @ 11, 5 say "Bemerkungen  : " + notiz

      @ 15, 5 say "'v' = vorwaerts   blaettern"
      @ 16, 5 say "'r' = rueckwaerts blaettern"
      @ 17, 5 say "'0' = zurueck ins Menue"    get antwort
      read

      do case

        case antwort = "v" .or. antwort = "V"
          if .not. eof
            skip
          endif
          if eof
            erase
            @  2, 5 say "Ende der Datei erreicht!"
            @  4, 5 say "Weiter mit bel. Taste    " get antwort
            read
            store " " to anfang
            store "0" to antwort
            enddo
            store " " to antwort
            enddo
          endif
          case antwort = "r" .or. antwort = "R"
            skip-1

          case antwort = "0"
            store " " to anfang
        endcase
  enddo
  store " " to antwort
enddo

use
release anfang
store " " to antwort
return
```

Listing 5. »ausgabe« übernimmt die formatierte Ausgabe auf den Bildschirm

```
* -------------------------------------------------------
* Listing Nr. 6
* -------
* DRUCKEN
* -------
* dient zum Ausdrucken von Adressenlisten oder Etiketten
* -------------------------------------------------------
*
store " " to antwort
store "A" to anfang
store "Z" to ende

do while antwort<>"l" .and. antwort<>"L" .and. antwort<>"e";
                     .and. antwort<>"E" .and. antwort<>"0"
  erase
  @  2, 5 say "'l' = Adressenliste drucken"
  @  3, 5 say "'e' = Etiketten     drucken"
  @  4, 5 say "'0' = zurueck ins Menue"
  @  7, 7 say "Druecken Sie den entsprechenden Buchstaben! " get antwort
  read

  if antwort="l" .or. antwort="L" .or. antwort="e" .or. antwort="E"
     @ 10, 5 say "Von Buchstabe " get anfang
     @ 11, 5 say "Bis Buchstabe " get ende
     read
     store !(anfang) to anfang
     store !(ende)   to ende
  endif
enddo

do case

  case antwort = "0"
    store " " to antwort
    release anfang
    release ende
    release nummer
    return

  case antwort="l" .or. antwort="L"
      use adressen index name
      locate for $(name,1,1)>=anfang
      set print on
      do while .not. eof
        ? " "
        ? " "
        ? " "
        ? "     ADRESSENLISTE (alph.)        Stand: " + date()
        ? "     ---------------------------------------------"
        ? " "
        store 1 to nummer

          do while .not. eof .and. nummer <= 20
            if $(name,1,1)>=anfang .and. $(name,1,1)<=ende
            ? "      " + $(anrede,1,4) + " " + trim(vorname) + " "
            ??  trim(name) + ", geb.: " + gebdat
            ? "            " + trim(strasse) + ", " + trim(plzort)
            ??  ", Tel.: " + telefon
            ? "            " + notiz
            store nummer + 1 to nummer
            endif

            skip
          enddo
        eject
       enddo
     set print off
     use
   endif

  case antwort="e" .or. antwort="E"
     use adressen index name
     locate for $(name,1,1)>=anfang
     set print on

     do while .not. eof
        if $(name,1,1)>=anfang .and. $(name,1,1)<=ende
           ? " "
           ? " "
           ? " "
           ? "  " + anrede
           ? "  " + trim(vorname) +  " " + !(name)
           ? "  " + strasse
           ? "  " + plzort
           ? " "
           ? " "
        endif

           skip
        enddo
      eject
      enddo
    set print off
    use
  endif

release all
return
```

Listing 6. Durch dieses Unterprogramm können Adreßlisten oder Etiketten gedruckt werden



CP/M – Programmieren mit Z80-Code

Mit dem UVMAC steht für den CP/M-Modus des C128 ein Z80-Assembler zur Verfügung, der sich besonders für Einsteiger eignet. Wir haben den UVMAC für Sie getestet und zeigen Ihnen, wie man damit arbeitet.

Das Betriebssystem CP/M wurde von Digital Research ursprünglich für den 8080-Prozessor entwickelt und wurde später unverändert an den Z80 angepaßt. Darin ist auch die Begründung für die beiden 8080-Assembler auf den Utility-Disketten zu suchen, die von der Firma D.I.S. angeboten werden. Da im C128 ein Z80-Prozessor eingebaut ist, benötigt man einen eigenen Z80-Assembler, um die Leistungsfähigkeit dieses Prozessors bei der Assembler-Programmierung voll ausnutzen zu können. Um die Unterschiede zwischen den beiden Assemblern zu verdeutlichen, wird ein Vergleich zwischen dem 8080- und dem Z80-Assembler gezogen.

Wegen der Komplexität des Z80 ist es für den Einsteiger ohne Assembler-Kenntnisse oft sehr schwierig, sich in den Z80 einzuarbeiten. Der Z80 besitzt über 1000 mögliche Befehle. Um den Einstieg in die Sprache des in Verwaltung und Industrie weit verbreiteten Z80 zu erleichtern, benötigt man ein Werkzeug, das sich leicht bedienen läßt. Der UVMAC erfüllt diese Anforderung und ist dadurch für die ersten Schritte mit dem Z80 geeignet. So können Makros ohne umständliche Definitionen einfach über die Anweisung »MAKRO« erzeugt werden. Um auch in Assembler strukturiert zu programmieren, bietet der UVMAC einen IF..ENDIF-Pseudo-Opcode. Für diese Schleifenstruktur steht sogar ein ELSE-Zweig zur Verfügung. Schließlich bietet UVMAC noch einige Pseudo-Opcodes zur Gestaltung der Listing-Ausgabe.

Bei der Assemblierung selbst können mehrere Ausgabedateien deklariert werden. Sie können zum Beispiel ein Listing in eine Datei ausgeben, ebenso die Fehlermeldungen. Der Programmierer sollte diese Optionen auf jeden Fall verwenden, da sonst nur die assemblierte »COM«-Datei auf Diskette ausgegeben wird. Die Fehlermeldungen erscheinen dann nur auf dem Bildschirm und sind so schnell wieder verloren.

Wie bereits erwähnt, eignet sich UVMAC sehr gut für den Einsteiger. Trotz der Komplexität des Z80-Prozessors läßt sich der UVMAC sehr einfach handhaben und bietet so dem Anwender die Möglichkeit, sich schnell in die Prozessorsprache einzuarbeiten.

Während der 8080-Assembler noch mit vielen verschiedenen Mnemonics arbeitet, die für das Verschieben von Speicher- und Registerinhalten verantwortlich sind, arbeitet der Z80 hier nur noch mit einem einzigen Befehl: »LD«. Dieser »versteht« allerdings dann eine Unzahl von verschiedenen Variationen. Der LD-Befehl ermöglicht alle Arten von Speicher- und Registermanipulation. Alle Adressierungsarten und auch sämtliche Ladebefehle, ob nun 8 Bit oder 16 Bit, werden über diesen leistungsstarken Befehl abgewickelt. Hierin liegt der gravierendste Unterschied zum 8080-Assembler. Um die Neuerungen des Z80 gegenüber dem 8080 etwas herauszustellen, sehen wir uns zwei Beispielprogramme an. Eines ist in Z80-Assembler (Listing 1), das andere in 8080-Assembler (Listing 2) geschrieben. Die Funktion beider Programme ist absolut identisch. Erst wird ein Text ausgegeben, dann eine Eingabe eingelesen und zum Schluß ein Text mit der Eingabe ausgegeben.

Alle Befehle zum Manipulieren von Speicherbereichen und Registerinhalten sind im 8080-Assembler mit verschiedenen Befehlen dargestellt, während beim Z80 alles über den LD-Befehl abgewickelt wird.

Die Pseudo-Opcodes, die für die Definition der Textkonstanten und der Variablen verwendet werden, sind nicht prozessor- sondern assemblerabhängig. Der wichtigste Unterschied zwischen den beiden Prozessoren besteht in der Anzahl der möglichen Befehle. Der Z80 versteht auf Grund einer speziellen Technik ein Vielfaches mehr an Befehlen, als beispielsweise der 8080 oder der im C64 verwendete 6502. Wenn Sie mehr über die Technik und den Aufbau des Z80-Prozessors erfahren wollen, finden Sie in diesem Sonderheft einen eigenen Z80-Artikel, der sich speziell mit diesen Problemen beschäftigt.

Der UVMAC eignet sich wegen der leichten Bedienung besonders für den Z80-Einsteiger, dem damit der Einstieg in die Welt des Z80-Prozessors erleichtert wird. Für Profis erweist sich das Produkt als nicht ausreichend, da der Assembler nicht relokatibel ist.

Der Assembler kostet inklusive Anleitung, die sehr knapp bemessen, aber ausreichend ist, 99 Mark. (rf)

Tesco GmbH, Rüdenhausenerstr., 8714 Wiesentheid
D.I.S. GmbH, Friedrich-Menze-Fricke-Str. 16-18, 4804 Versmold

```
;     Bildschirmausgabe unter CP/M mit Z80-Code
;
      ORG 100H                      ;Programmstart
BDOS  EQU 5                         ;BDOS-Einsprungadresse
CCP   EQU 0                         ;CP/M-Warmstart
OUT   EQU 9                         ;String-Ausgabe Routine
IN    EQU 10                        ;Puffer-Lese Routine
      LD  C,OUT
      LD  DE,P1
      CALL  BDOS                    ;Ausgabe Textkonstante P1
      LD  C,IN
      LD  DE,EINGABE
      CALL  BDOS                    ;Lesen Tastaturpuffer
      LD  C,OUT
      LD DE,P2
      CALL BDOS                     ;Ausgeben Textkonstante P2
      LD D,0                        ;Aufbereiten Eingabe
      LD A,(EINGABE+1)              ;Feststellen der genauen Textlaenge in EINGABE
      LD E,A                        ;In EINGABE+1 steht die wirkliche Laenge des Feldes
      LD HL,EINGABE
      ADD HL,DE                     ;Positionieren hinter letztem Zeichen
      INC HL                        ;in EINGABE
      INC HL
      LD A,'$'                      ;Endekennzeichen in EINGABE setzen
      LD (HL),A
      LD C,OUT
      LD DE,EINGABE
      CALL BDOS                     ;Ausgeben von EINGABE
      JP CCP                        ;Zurueck zu CP/M
      ;
P1:   DEFB 'Ihren Namen bitte: $'
P2:   DEFB 0DH,0AH,0AH,'CP/M gruesst Sie herzlich - $'
EINGABE:
      DEFB 25
      DEFS 26
      END
```

Listing 1. Demo-Programm in Z80-Quellprogramm

```
      ORG 100H
BDOS  EQU 5                   ;BDOS-Einsprungadresse
CCP   EQU 0                   ;Warmstart
PRT   EQU 9                   ;String-Ausgaberoutine
INP   EQU 10                  ;Puffer-Lese-Routine
      MVI C,PRT
      LXI D,P1
      CALL BDOS                    ;Ausgabe Textkonstante P1
      MVI C,INP
      LXI D,EINGABE
      CALL BDOS                    ;Lesen Tastaturpuffer
      MVI C,PRT
      LXI D,P2
      CALL BDOS                    ;Ausgeben Textkonstante P2
      MVI D,0                      ;Aufbereiten Eingabe
      LDA EINGABE+1                ;Feststellen der genauen Textlaenge in Eingabe
      MOV E,A                      ;In EINGABE+1 steht die wirkliche Laenge des Feldes
      LXI H,EINGABE
      DAD D                        ;Positionieren hinter letztem Zeichen
      INX H
      INX H
      MVI A,'$'                    ;Endekennzeichen in EINGABE setzen
      MOV M,A
      MVI C,PRT
      LXI D,EINGABE
      CALL BDOS                    ;Ausgeben von EINGABE
      JMP CCP                      ;Zurueck zu CP/M
;
P1 DB 'Ihren Namen bitte: $'
P2 DB 0DH,0AH,0AH,'CP/M gruesst Sie herzlich - $'
EINGABE DB 25
        DS 26
END
```

Listing 2. Dasselbe Programm in 8080-Quellcode

Datenaustausch zwischen CP/M und C 64/C 128

Mit dem vollständig in Maschinensprache geschriebenen Programm »CP/M<-> CBM« wird zum ersten Mal die Möglichkeit geboten, Daten und Programme auf einfachem Weg zwischen dem CP/M- und C 64-Format beliebig hin und her zu transferieren. Weiterhin kann der C 64 nun Disketten im CP/M 2.2 oder 3.0-Format erzeugen.

Durch die Zwittergestalt des C 128 - er ist C 64- und CP/M-Computer gleichzeitig - kommt es vor, daß man seine eigenen Dateien nicht in jedem Modus zur Verfügung hat. Der gleiche Effekt tritt ein, wenn man den C 64 mit einem CP/M-Modul ausstattet. Es wird also ein Programm benötigt, das diesem Mißstand abhilft. Ein Programm dieser Art hat vielfältige Anwendungsmöglichkeiten. An dieser Stelle seien nur einige der wichtigsten exemplarisch aufgezählt:

- Unter CP/M kann weder der C 64 noch der C 128 ohne Hardware-Erweiterungen an der Datenfernübertragung teilnehmen. Was aber, wenn Ihnen Ihr Partner an der Telefonleitung sein neuestes Turbo-Pascal-Programm schicken möchte? Ohne das Programm »CP/M<-> CBM« (Listing 1) muß er Ihnen eine Diskette senden, die Sie, wenn nicht der gleiche Computer verwendet wird, oft nicht einmal lesen können.
- Sie möchten von Ihrem xyz-Pascal auf ein Pascal unter CP/M umsteigen. Bis jetzt gab es keine Möglichkeit, Ihre Quelldateien wiederverwenden zu können, ohne sie neu eintippen zu müssen. Mit »CP/M<->CBM« ist dies kein Problem mehr! Gleiches gilt übrigens auch für Basic. Ihre wertvollen Programme können nach meist kleinen Änderungen auch unter Microsoft- oder C-Basic laufen. Und da Microsoft-Basic sowohl umfangreicher als Standard-Basic als auch auf allen CP/M-Computern lauffähig ist, werden Ihre Programme für einen viel größeren Anwenderkreis wertvolle Hilfsmittel.
- C 64-Anwender können endlich auch CP/M-3.0-Disketten des C 128 lesen, wenn diese einseitig beschrieben wurden. Der C 128 wiederum kann beide Formate, CP/M 2.2 und CP/M 3.0, verarbeiten.
- Sie möchten Texte, etwa Assembler-Listings, mit einem guten 80-Zeichen-Textverarbeitungssystem bearbeiten. Mit dem Programm »CP/M<->CBM« ist es kein Problem, diese Dateien vom C 64 nach CP/M zu transferieren. Unter diesem Betriebssystem stehen Ihnen dann sehr leistungsfähige Textverarbeitungssysteme zur Verfügung.
- Obwohl die Preise für Disketten stark gefallen sind, ist das Speichermedium Kassette noch immer billiger. Daher werden noch immer manche Sicherheitskopien auf Kassette abgelegt. Dies ist nun auch für CP/M-Dateien möglich.

Wie man an diesen wenigen Beispielen sieht, ist das Anwendungsgebiet für »CP/M<->CBM« sehr groß. Nicht umsonst wurde eine Menge Zeit in die Entwicklung und in das Austesten investiert. Programmierer, die in Assembler arbeiten, wissen was es heißt, ein über sechs KByte (26 Blocks) langes Maschinenprogramm mit einem Assemblerquelltext von über 78K (313 Blocks) zu schreiben.

Schalten Sie den C 64 (oder C 128 im C 64-Modus), die Floppy und Ihren Monitor ein und legen Sie die Diskette mit dem Programm »CP/M<->CBM« in das Laufwerk 8. Das Programm wird nun mit

```
LOAD "CP/M<->CBM",8
```

gefolgt von <RETURN> geladen und mit RUN <RETURN> gestartet. Sie können nun die Diskette aus dem Laufwerk nehmen. Alle weiteren Eingaben sind durch den Programmaufbau selbsterklärend. Eingabefehler werden vom Programm abgefangen und durch ein akustisches Signal angezeigt. Zur leichteren Benutzung ist das Programm vollkommen menügesteuert, ohne aber durch Unter- und Unteruntermenüs schwerfällig zu werden.

Nur durch die Funktionstasten lassen sich die Menüpunkte

Programmbedienung

-1- Directory CP/M,
-2- Directory C 64,
-3- Kopieren von CP/M nach C 64,
-4- Kopieren von C 64 nach CP/M,
-5- Löschen der CP/M-Directory (entspricht Formatieren ohne ID),
-6- Formatieren im CP/M-2.2-Format und
-7- Formatieren im CP/M-3.0-Format

anwählen. An den Stellen, an denen Daten verloren gehen könnten, fragt das Programm noch einmal bei dem Benutzer nach. Für die Auswahl der zu transferierenden Dateien stehen zwei Wahlmöglichkeiten zur Verfügung. Die erste erlaubt für jede Datei einzeln, verschiedene Parameter festzulegen, bei der zweiten Möglichkeit erfolgt die Eingabe blockorientiert. Bei keiner der beiden Möglichkeiten ist der Benutzer gezwungen, Namen oder ähnliche, in (Tip)-Arbeit ausartende, Eingaben zu machen. »J« und »N« beziehungsweise <RETURN> für standardmäßig vorgewählte Angaben reichen vollkommen aus. Standardantworten sind dabei durch reverse Zeichen hervorgehoben. Bei der Blockauswahl besteht nebenbei die Möglichkeit, die Programme auf der Zieldiskette in einer anderen Reihenfolge als auf der Quelldiskette anzuordnen. Die Eingabe

3-7,2,14-

würde beispielsweise bewirken, daß zuerst die dritte bis siebte, dann die zweite und schließlich alle Dateien, von der vierzehnten an, transferiert würden. Arbeiten Sie mit einem Laufwerk, wird das Programm Sie auffordern, die Disketten zu wechseln. Beim Arbeiten mit zwei Laufwerken gibt das Programm nur den Namen der Dateien aus, die es gerade abarbeitet. »CP/M<->CBM« erkennt Speicherplatzüberlauf im RAM und auf Diskette, ebenso ein Überlaufen des Directory und teilt dies dem Benutzer in deutschem Klartext mit. Eine kleine Ausnahme machen die Fehlermeldungen des Laufwerkes. Sie werden direkt übernommen und sind somit in Englisch abgefaßt. Wie allgemein bekannt ist, gibt es kein perfektes Programm. Um keine falschen Vorstellungen aufkommen zu lassen, soll auch auf die Grenzen dieses Programmes hingewiesen werden. Mit dem Programm »CP/M<->CBM« können keine Dateien von Disketten kopiert werden, die im 1571-doppelseitigen Modus beschrieben wurden. Diese Einschränkung wurde von uns in Kauf genommen, da die 1571 340 KByte Speicherplatz im CP/M-Modus zur Verfügung stellt. Alle Verwaltungen müßten dadurch mit 16-Bit-Zahlen durchgeführt werden, was im Gegensatz zu der verwendeten 8-Bit-Organisation eine Verlangsamung des Programms auf etwa die Hälfte der aktuellen Geschwindigkeit bedeuten würde, da statt der Prozessorre-

gister RAM-Adressen benutzt werden müßten. Weiterhin war, als mit der Programmentwicklung auf dem C 64 begonnen wurde, auf dem Markt noch lange keine Floppy 1571 erhältlich. Aus den Handbüchern, die dem C 128 beiliegen, ist über die Verwaltung der Directory des C 64 oder unter CP/M so gut wie nichts ersichtlich. Das gesamte »Deblocking«, das ist die Zuordnung der physikalischen Tracks und Sektoren zu den logischen Records, mußte in mühsamer Kleinarbeit ausgeknobelt werden. Des weiteren kann das Programm keine beliebig langen Dateien transferieren. Es wurde bei diesem Programm vor allem auf Kompatibilität Wert gelegt. Aus diesem Grund werden auch, bis auf eine dokumentierte Ausnahme, nur Kernelroutinen verwendet. Das Programm sollte schließlich auch dem C 64-Besitzer weiterhelfen und mußte somit auf dem C 64 wie auf dem C 128 (im C 64-Modus) lauffähig sein. Dies führt dazu, daß der Speicherplatz, den der C 128 bietet, nicht ausgenutzt wird, da das Programm nur im C 64-Modus läuft. Dieser Nachteil wird durch die Arbeitsgeschwindigkeit, die in einem »nichtgebankten« System viel höher ist als in einem gebankten, bei weitem wieder wettgemacht. Außerdem sind etwa 40 KByte (zirka 160 Blocks), die transferiert werden können, mehr als in den meisten Fällen benötigt werden. Hierbei wird das Basic-ROM vom Programm weggeschaltet, was wohl niemanden überrascht, der sich ein wenig in der »Speicherlandschaft« auskennt. Die maximale Anzahl an Programmnamen auf einer Diskette wurde mit 80 festgelegt, was für CP/M-Disketten ausreicht.

Grenzen des Programmes

»CP/M <->CBM« unterstützt keine »Stamps« (Password, Create, Update, Access etc.). Dies heißt aber nicht, daß Fehler entstehen, wenn auf eine Diskette kopiert wird, die bereits Programme mit solchen Stamps enthält. »CP/M <->CBM« läßt die Stamps mit ihren Einträgen einfach unberücksichtigt. Auch wird bei dem Kopieren von C 64 nach CP/M nicht geprüft, ob sich bereits ein Programm gleichen Namens auf der Zieldiskette befindet. Sind C 64-Namen länger als acht Buchstaben, Punkt und drei Buchstaben Extention, werden die Namen auf die ersten acht Buchstaben des jeweiligen Namens gekürzt. Fehlt der Punkt, werden als Extention drei Leerzeichen eingesetzt. Ist die Extention länger als drei Buchstaben, wird sie auf die ersten drei Buchstaben gekürzt. Auch hier wird nicht geprüft, ob sich durch das Kürzen der Namen gleiche Namen ergeben. Auf diesen Punkt müssen Sie als Benutzer achten! Verwenden Sie im C 64-Modus also gültige CP/M-Namen, so ist die zweite der angeführten Fehlermöglichkeiten ausgeschaltet. Auf zwei weitere Einschränkungen sei noch hingewiesen:

- Es können keine relativen Dateien kopiert werden.
- Probleme treten möglicherweise auch bei commodorespezifischen Zeichen, wie beispielsweise Grafikzeichen, Cursor-Steuerzeichen und ähnlichem auf, denn in was sollte das Programm diese Zeichen umcodieren?

Wenn beim Kopieren keine Umcodierung gewählt wurde, werden alle 256 möglichen Zeichen direkt übertragen. Somit können mit diesem Programm ohne weiteres auch Maschinenprogramme transferiert werden. Natürlich ist, aus verständlichen Gründen, dieses Programm nicht in der Lage, Z80-Maschinenprogramm auf den 6502 umzucodieren. Eine kleine Eigenheit steckt in der Wahl des C 64-Dateityps. Ist eine Datei vom Typ »PRG« oder wird als Ziel eine »PRG«-Datei gewünscht, wird beim Lesen der Dateien die Ladeadresse überlesen und beim Schreiben die Ladeadresse $1100 vor der Datei ausgegeben. Dies ist sinnvoll, da das Betriebssystem diese beiden Bytes als Anfangsadresse beim Laden interpretieren und unterschlagen würde, wenn Sie die Datei mit LOAD oder dem entsprechenden Monitorbefehl in den Computer einladen wollten. Speichern Sie einen RAM-Bereich ab, werden diese Bytes automatisch mit ausgegeben, damit Ihnen keine Informationen verloren gehen, beziehungsweise Sie keine zwei Byte zuviel in den übertragenen Dateien wiederfinden. Das könnte unter CP/M unerwünschte Effekte hervorrufen.

Wie Sie sehen, stecken in diesem relativ kleinen Programm doch einige Fähigkeiten, die Sie auf einem C 64 oder einem C 128 im C 64-Modus, mit einem oder zwei Laufwerken, mit oder ohne Floppybeschleuniger, voll ausschöpfen können. »CP/M <->CBM« wird Ihnen schnell ein unentbehrliches Hilfsmittel werden und hilft sicherlich, einige Brücken zu neuen Anwendungsmöglichkeiten zu schlagen, nicht zuletzt auch der Tatsache wegen, daß der C 128 wiederum das Datenformat des C 64 lesen kann, wodurch eine universelle Schnittstelle geschaffen ist. (D. Winkler/bj)

```
Name : cp/m<->cbm           0801 21d7
-------------------------------------
0801 : 24 08 c1 07 9e 28 32 30  cf
0809 : 38 36 29 20 20 57 52 49  43
0811 : 54 54 45 4e 20 42 59 20  64
0819 : 12 22 cb d2 d7 20 26 20  e1
0821 : d4 d7 00 00 00 a9 36 85  12
0829 : 01 20 da 0e 20 90 09 a9  b1
0831 : 69 a0 08 20 cf 09 20 e4  80
0839 : ff f0 fb a2 07 dd 51 08  b8
0841 : f0 05 ca 10 f8 30 ef bd  b5
0849 : 59 08 48 bd 61 08 48 60  a8
0851 : 85 86 87 88 89 8a 8b 58  d8
0859 : 0f 0c 11 18 20 21 21 0a  59
0861 : 77 ba 70 72 79 0a 0d 1b  f2
0869 : 09 0e 08 0d 0d 20 20 20  b0
0871 : 20 2d 20 c6 31 20 2d 20  12
0879 : 20 20 20 c4 49 52 45 4b  1d
0881 : 54 4f 52 59 20 28 c3 d0  31
0889 : 2f cd 29 0d 0d 20 20 20  1d
0891 : 20 2d 20 c6 33 20 2d 20  52
0899 : 20 20 20 c4 49 52 45 4b  3d
08a1 : 54 4f 52 59 20 28 c3 c2  34
08a9 : cd 29 0d 0d 20 20 20 20  b4
08b1 : 2d 20 c6 35 20 2d 20 20  73
08b9 : 20 20 c3 d0 2f cd 20 2d  30
08c1 : 3e 20 c3 c2 cd 0d 0d 20  12
08c9 : 20 20 20 2d 20 c6 37 20  fc
08d1 : 2d 20 20 20 20 c3 c2 cd  e1
08d9 : 20 2d 3e 20 c3 d0 2f cd  3f
08e1 : 0d 0d 20 20 20 20 2d 20  79
08e9 : c6 32 20 2d 20 20 20 20  3a
08f1 : c3 d0 2f cd 2d c4 49 52  65
08f9 : 45 4b 54 4f 52 59 20 4c  ec
0901 : 4f 45 53 43 48 45 4e 0d  32
0909 : 0d 20 20 20 20 2d 20 c6  ac
0911 : 34 20 2d 20 20 20 20 c4  b2
0919 : 49 53 4b 20 46 4f 52 4d  a5
0921 : 41 54 49 45 52 45 4e 20  50
0929 : c3 d0 2f cd 20 32 2e 32  8b
0931 : 0d 0d 20 20 20 20 2d 20  c9
0939 : c6 36 20 2d 20 20 20 20  8c
0941 : c4 49 53 4b 20 46 4f 52  fe
0949 : 4d 41 54 49 45 52 45 4e  0e
0951 : 20 c3 d0 2f cd 20 33 2e  74
0959 : 30 0d 0d 0d 20 20 20 20  b9
0961 : 2d 20 20 58 20 2d 20 20  de
0969 : 20 20 c5 4e 44 45 20 2d  1e
0971 : 3e 20 c2 c1 d3 c9 c3 0d  5d
0979 : 0d 20 20 20 20 2d d3 d4  07
0981 : cf d0 2d 20 20 20 20 c1  0f
0989 : 42 42 52 55 43 48 00 a9  f5
0991 : 97 a0 09 4c cf 09 93 20  18
0999 : 20 20 20 20 20 20 20 20  99
09a1 : 20 20 c3 d0 2f cd 20 3c  37
09a9 : 2d 3e 20 c3 c2 cd 0d 20  85
09b1 : 20 20 20 20 20 28 43 29  90
09b9 : 20 42 59 20 cd 41 52 4b  1b
09c1 : 54 20 26 20 d4 45 43 48  c8
09c9 : 4e 49 4b 0d 0d 00 85 6d  f2
09d1 : 84 6e a0 00 b1 6d d0 01  80
09d9 : 60 20 d2 ff e6 6d d0 f2  01
09e1 : e6 6e 4c d3 09 48 8a 48  19
09e9 : a2 0f 8e 18 d4 a2 27 8e  d6
09f1 : 00 d4 a2 34 8e 01 d4 a2  14
09f9 : 00 8e 05 d4 a2 f9 8e 06  5c
0a01 : d4 a2 21 8e 04 d4 a2 00  b2
0a09 : 8a e8 d0 fd 18 69 04 d0  7a
0a11 : f8 8e 04 d4 8e 18 d4 68  ba
0a19 : aa 68 60 20 da 0e a2 fa  b2
0a21 : 9a a9 37 85 01 4c 9d e3  bf
0a29 : 48 a5 91 c9 7f f0 02 68  3a
0a31 : 60 a2 fb 9a a9 55 a0 0a  10
0a39 : 20 cf 09 20 e6 09 20 da  74
0a41 : 0e a9 82 a0 0a 20 cf 09  cb
0a49 : 20 e4 ff f0 fb c9 03 f0  f5
0a51 : f7 4c 2d 08 0d 0d 12 20  7d
0a59 : 3f 20 4d 49 54 20 d3 d4  64
0a61 : cf d0 20 41 42 47 45 42  c1
0a69 : 52 4f 43 48 45 4e 20 0d  9e
0a71 : 00 a9 82 a0 0a 20 cf 09  ed
0a79 : 20 29 0a 20 e4 ff f0 f8  b8
0a81 : 60 11 20 20 20 20 20 20  3a
0a89 : 20 20 20 20 20 20 20 20  89
0a91 : 12 20 d4 41 53 54 45 20  3e
0a99 : 92 00 20 da 0b 20 e0 0b  da
0aa1 : ae af 02 a0 0f 20 ba ff  98
0aa9 : a9 01 a2 ba a0 0a 20 bd  29
0ab1 : ff 20 c0 ff b0 04 4c 89  60
0ab9 : 0b 49 c9 05 f0 06 20 da  f1
0ac1 : 0e 4c 2d 08 a9 cc a0 0a  d9
0ac9 : 4c eb 0a 0d 0d 12 20 3f  8f
0ad1 : 20 c4 52 49 56 45 20 41  a4
```

Listing 1. »CP/M <-> CBM«. Bitte mit dem MSE eingeben. (Anfang)

```
Ød09 : ad 12 29 8d a8 02 a2 02  e4
```

```
Øad9 : 4e 54 57 4f 52 54 45 54  97
Øae1 : 20 4e 49 43 48 54 20 Ød  a5
Øae9 : Ød 00 a2 fb 9a 8d b4 02  Øb
Øaf1 : 8c b2 02 20 cc ff 20 da  5e
Øaf9 : Øe ad b4 02 ac b2 02 20  f4
Øb01 : cf 09 20 e6 09 20 72 Øa  a9
Øb09 : 4c 2d 08 20 d4 Øb 20 e7  e8
Øb11 : Øb ae af 02 a8 20 ba ff  16
Øb19 : a9 01 a2 29 a0 Øb 20 bd  6f
Øb21 : ff 20 c0 ff 90 90 b0 92  d6
Øb29 : 23 8d b4 02 8e b0 02 8c  10
Øb31 : b2 02 20 67 20 20 d4 Øb  46
Øb39 : 20 e7 Øb ae af 02 a8 20  d3
Øb41 : ba ff ae b0 02 ac b2 02  11
Øb49 : ad b4 02 20 bd ff 20 c0  b3
Øb51 : ff b0 03 4c 89 Øb c9 05  15
Øb59 : d0 03 4c c5 Øa c9 04 f0  57
Øb61 : 03 4c 89 Øb a9 6c a0 Øb  e5
Øb69 : 4c eb Øa Ød Ød 12 20 3f  2f
Øb71 : 20 c6 49 4c 45 20 4e 49  f1
Øb79 : 43 48 54 20 47 45 46 55  5c
Øb81 : 4e 44 45 4e 20 Ød Ød 00  ab
Øb89 : 20 cc ff 20 e0 Øb aa 20  65
Øb91 : c6 ff a2 05 bd ce Øb 9d  5a
Øb99 : 3c 03 ca 10 f7 a2 00 8e  bd
Øba1 : b8 02 20 cf ff ae b8 02  b9
Øba9 : 9d 42 03 ee b8 02 c9 Ød  e3
Øbb1 : d0 f0 a9 00 e8 9d 42 03  ee
Øbb9 : 20 cc ff ad 42 03 c9 32  bd
Øbc1 : b0 01 60 20 90 09 a9 3c  7e
Øbc9 : a0 03 4c eb Øa Ød Ød 12  dd
Øbd1 : 20 3f 20 20 e7 Øb 4c c3  2c
Øbd9 : ff 20 e0 Øb 4c c3 ff 18  95
Øbe1 : ad af 02 69 06 60 38 ad  b3
Øbe9 : af 02 e9 06 60 8e 79 Øc  4d
Øbf1 : 85 69 84 6a 20 20 20 20  5d
Øbf9 : e7 Øb 09 30 8d 77 Øc e0  35
Øc01 : 31 d0 1f 20 48 Øc 20 e7  9b
Øc09 : Øb aa 20 c6 ff a0 00 8c  68
Øc11 : b8 02 20 cf ff ac b8 02  19
Øc19 : 91 69 ee b8 02 d0 f3 4c  41
Øc21 : cc ff 8d b8 Øc 20 92 Øc  8b
Øc29 : 20 e7 Øb aa 20 c9 ff a0  e7
Øc31 : 00 8c b8 02 ac b8 02 b1  e2
Øc39 : 69 20 d2 ff ee b8 02 d0  c5
Øc41 : f3 20 48 Øc 4c cc ff a9  56
Øc49 : 06 8d b9 02 20 e0 Øb aa  4f
Øc51 : 20 c9 ff ae b9 02 bd 74  b7
Øc59 : Øc 20 d2 ff ce b9 02 10  Ød
Øc61 : f2 ad a7 02 20 7b Øc 20  a2
Øc69 : bd Øe ad a8 02 20 7b Øc  d5
Øc71 : 4c cc ff 20 30 20 58 20  cd
Øc79 : 58 55 a2 00 38 e9 Øa 90  41
Øc81 : 03 e8 d0 f9 69 Øa 48 8a  89
Øc89 : 20 8d Øc 68 09 30 4c d2  69
Øc91 : ff 20 e0 Øb aa 20 c9 ff  Ød
Øc99 : a2 ff 8e b8 02 ee b8 02  74
Øca1 : ae b8 02 e0 07 b0 09 bd  de
Øca9 : b4 Øc 20 d2 ff 4c 9e Øc  bb
Øcb1 : 4c cc ff 42 2d 50 20 58  32
Øcb9 : 20 30 20 da Øe a9 14 20  13
Øcc1 : 50 Øe 20 cf Øc 20 da Øe  64
Øcc9 : 20 72 Øa 4c 2d 08 20 Øa  d6
Øcd1 : Øf 20 73 Øe 8d ad 02 8d  f8
Øcd9 : af 02 a9 12 8d a7 02 a9  a7
Øce1 : 01 8d a8 02 8d ba 02 20  Øa
Øce9 : 90 09 20 9b Øa 20 Øc Øb  61
Øcf1 : 20 29 Øa a9 11 a0 29 a2  5e
Øcf9 : 31 20 ee Øb ad 11 29 c9  f3
Ød01 : 24 90 02 a9 00 8d a7 02  32
Ød09 : ad 12 29 8d a8 02 a2 02  e4
Ød11 : 8e b8 02 a2 08 8e b9 02  b0
Ød19 : ae bf 02 20 b6 Ød a0 00  82
Ød21 : ae b8 02 bd 11 29 d0 Øe  1d
Ød29 : 8a 18 69 20 8d b8 02 ce  62
Ød31 : b9 02 d0 e4 f0 43 91 6b  02
Ød39 : e8 e8 e8 c8 bd 11 29 91  15
Ød41 : 6b c0 12 90 f5 ee bf 02  7d
Ød49 : ad bf 02 c9 50 90 1c a9  dd
Ød51 : 57 a0 Ød 4c eb Øa Ød Ød  23
Ød59 : 12 20 3f 20 d0 55 46 46  ad
Ød61 : 45 52 20 56 4f 4c 4c 20  6b
Ød69 : Ød Ød 00 18 ad b8 02 69  7b
Ød71 : 20 8d b8 02 ce b9 02 d0  2b
Ød79 : 9f ad a7 02 f0 03 4c f1  55
Ød81 : Øc 20 d4 Øb ae bf 02 ca  ba
Ød89 : d0 17 a9 92 a0 Ød 4c eb  1d
Ød91 : Øa Ød Ød 12 20 c4 49 53  9b
Ød99 : 4b 20 4c 45 45 52 20 Ød  32
Øda1 : 00 a2 01 20 8a 16 20 d5  bc
Øda9 : Øe 20 57 Øe ae ba 02 ec  02
Ødb1 : bf 02 90 ef 60 a9 21 85  76
Ødb9 : 6c a9 d7 ca f0 09 18 69  d3
Ødc1 : 13 90 f8 e6 6c d0 f4 85  63
Ødc9 : 6b 60 ad ba 02 ee ba 02  ad
Ødd1 : a2 00 8d c6 02 8e c7 02  67
Ødd9 : a9 00 a2 05 9d c7 02 ca  91
Øde1 : d0 fa 38 fe c8 02 ad c6  fd
Øde9 : 02 fd 46 Øe 8d c6 02 ad  b0
Ødf1 : c7 02 fd 4b Øe 8d c7 02  13
Ødf9 : b0 e9 de c8 02 ad c6 02  1b
Øe01 : 7d 46 Øe 8d c6 02 ad c7  99
Øe09 : 02 7d 4b Øe 8d c7 02 e8  4f
Øe11 : e0 05 90 ce a2 00 bd c8  24
Øe19 : 02 d0 13 a9 20 8e b1 02  bf
Øe21 : 20 d2 ff ae b1 02 e8 e0  11
Øe29 : 04 90 eb bd c8 02 8e b1  62
Øe31 : 02 20 8d Øc ae b1 02 e8  7b
Øe39 : bd c8 02 e0 05 90 ef 20  cc
Øe41 : 29 Øa 4c bd Øe 10 e8 64  08
Øe49 : Øa 01 27 03 00 00 00 8d  19
Øe51 : b7 02 8d b6 02 60 48 ce  25
Øe59 : b6 02 d0 14 ad b7 02 8d  83
Øe61 : b6 02 8a 48 98 48 20 72  f5
Øe69 : Øa 20 90 09 68 a8 68 aa  8b
Øe71 : 68 60 20 90 09 a9 9c a0  b5
Øe79 : Øe 20 cf 09 20 79 Øa f0  84
Øe81 : fb c9 Ød f0 Øb c9 38 f0  84
Øe89 : 07 c9 39 d0 ef a9 09 2c  a6
Øe91 : a9 08 48 20 8d Øc 20 d5  ba
Øe99 : Øe 68 60 Ød Ød 20 20 20  28
Øea1 : 20 20 20 20 cc 41 55 46  96
Øea9 : 57 45 52 4b 20 28 12 38  9d
Øeb1 : 92 2f 39 29 20 00 20 bd  4c
Øeb9 : Øe 20 bd Øe 8d b4 02 8e  ac
Øec1 : b0 02 8c b2 02 a9 20 20  1a
Øec9 : d2 ff ae b0 02 ac b2 02  b1
Øed1 : ad b4 02 60 a9 Ød 4c d2  3f
Øed9 : ff 20 cc ff 20 67 20 ad  34
Øee1 : ad 02 f0 Øe cd ae 02 f0  c9
Øee9 : Øe 8d af 02 20 d4 Øb 20  ff
Øef1 : da Øb ad ae 02 f0 09 8d  79
Øef9 : af 02 20 d4 Øb 20 da Øb  7f
Øf01 : a9 00 8d ad 02 8d ae 02  Øf
Øf09 : 60 a9 01 8d bf 02 20 67  8b
Øf11 : Øf a2 51 20 b6 Ød a5 6b  Øb
Øf19 : a6 6c 8d b8 02 8e b9 02  ef
Øf21 : a2 01 20 b6 Ød 38 ad b8  dd
Øf29 : 02 e5 6b 8d b8 02 ad b9  70
Øf31 : 02 e5 6c 8d b9 02 a0 00  21
Øf39 : a9 a0 91 6b e6 6b d0 02  15
Øf41 : e6 6c ce b8 02 d0 03 ce  3c
Øf49 : b9 02 10 ee a2 50 8e b8  3e
Øf51 : 02 ae b8 02 20 b6 Ød a9  58
Øf59 : 00 a0 11 91 6b c8 91 6b  3a
Øf61 : ce b8 02 d0 ec 60 a9 00  9e
Øf69 : a2 aa 9d 66 28 ca d0 fa  a7
Øf71 : ce 67 28 ce 68 28 60 20  60
Øf79 : da Øe a9 14 20 50 Øe 20  44
Øf81 : 8c Øf 20 da Øe 20 72 Øa  b8
Øf89 : 4c 2d 08 20 Øa Øf a9 00  32
Øf91 : 8d cd 02 20 73 Øe 8d ae  c5
Øf99 : 02 8d af 02 20 90 09 20  79
Øfa1 : ea Øf a2 00 8e be 02 a0  e4
Øfa9 : 04 8c c0 02 20 fa 16 a2  db
Øfb1 : 31 a9 11 a0 29 20 ee Øb  75
Øfb9 : 20 29 Øa 20 65 10 ce c0  88
Øfc1 : 02 f0 06 20 15 18 4c b0  66
Øfc9 : Øf ee be 02 ae be 02 e0  ea
Øfd1 : 02 90 d4 2c cd 02 10 01  05
Øfd9 : 60 ae bf 02 ca f0 03 4c  99
Øfe1 : a2 Ød a9 92 a0 Ød 4c eb  42
Øfe9 : Øa a2 01 8e a7 02 ca 8e  29
Øff1 : a8 02 20 9b Øa 20 Øc Øb  fe
Øff9 : a2 31 a9 11 a0 29 20 ee  72
1001 : Øb 2c cd 02 30 Øa a9 42  54
1009 : a0 10 20 cf 09 20 57 Øe  be
1011 : a2 02 bd 3f 10 dd 11 29  92
1019 : d0 06 ca 10 f5 a2 ff 2c  6e
1021 : a2 00 8e ce 02 2c cd 02  fd
1029 : 10 01 60 a9 5b a0 10 2c  5a
1031 : ce 02 10 04 a9 60 a0 10  c5
1039 : 20 cf 09 4c 57 Øe 43 42  84
1041 : 4d Ød 20 c4 49 53 4b 45  9c
1049 : 54 54 45 4e 46 4f 52 4d  a5
1051 : 41 54 3a 20 c3 d0 2f cd  6a
1059 : 20 00 32 2e 32 Ød 00 33  be
1061 : 2e 30 Ød 00 a2 08 8e b4  f9
1069 : 02 8e ba 02 a2 00 8e b9  79
1071 : 02 a0 10 8c bb 02 bd 11  3e
1079 : 29 f0 Øb 8a 18 69 20 aa  d1
1081 : ce ba 02 d0 e9 60 e8 8e  a9
1089 : b0 02 2c cd 02 30 Øc 20  11
1091 : 2d 16 8d b4 02 90 49 aa  e2
1099 : 20 b6 Ød ad b0 02 18 69  5c
10a1 : Øf aa bc 11 29 d0 23 2c  55
10a9 : cd 02 30 18 ad b0 02 18  1f
10b1 : 69 Øe aa bd 11 29 a0 11  83
10b9 : 18 71 6b 91 6b c8 a9 00  3a
10c1 : 71 6b 91 6b ad b9 02 4c  03
10c9 : 7d 10 b9 67 28 f0 03 20  00
10d1 : 20 11 ad b4 02 99 67 28  57
10d9 : e8 ce bb 02 d0 c4 f0 c7  de
10e1 : ae bf 02 20 b6 Ød ee bf  02
10e9 : 02 ae bf 02 e0 51 90 03  53
10f1 : 4c 50 Ød a0 01 ae b0 02  09
10f9 : bd 11 29 29 7f c9 20 d0  17
1101 : 02 a9 a0 91 6b e8 c8 c0  d5
1109 : 09 90 ed a9 2e 91 6b c8  ba
1111 : bd 11 29 29 7f 91 6b e8  ca
1119 : c0 Øc 90 f3 4c 9c 10 8d  87
1121 : b5 02 8e b1 02 8c b3 02  08
1129 : a9 56 a0 11 20 cf 09 ae  4a
1131 : bf 02 ca 20 b6 Ød a0 01  00
1139 : b1 6b 20 d2 ff c8 c0 Øe  68
1141 : 90 f6 20 d5 Øe 20 72 Øa  cf
1149 : 20 d5 Øe ad b5 02 ae b1  17
1151 : 02 ac b3 02 60 Ød Ød 12  9d
1159 : 20 3f 20 c2 45 4c 45 47  d4
1161 : 55 4e 47 53 46 45 48 4c  62
1169 : 45 52 20 49 4e 3a 20 00  40
1171 : 20 da Øe a9 ff 20 50 Øe  15
1179 : 20 8c Øf a9 00 8d cd 02  80
1181 : 20 a0 1c 20 90 09 a9 d7  a4
1189 : a0 13 20 cf 09 20 73 Øe  30
1191 : 8d ad 02 a0 ff 8c cf 02  31
1199 : c8 8c bb 02 cd ae 02 d0  d2
11a1 : 03 ee cf 02 2c cf 02 10  b9
11a9 : 16 a9 74 a0 13 20 cf 09  48
11b1 : ad ad 02 8d af 02 20 8d  Øe
11b9 : Øc 20 72 Øa 20 9b Øa 20  fb
11c1 : 90 09 a9 bf a0 13 20 cf  fb
11c9 : 09 ac bb 02 be c7 27 20  5e
11d1 : b6 Ød a2 00 a0 01 b1 6b  66
11d9 : c9 a0 f0 09 9d 5a 03 e8  da
11e1 : c8 c0 09 90 f1 a0 09 b1  09
11e9 : 6b c9 a0 f0 09 9d 5a 03  6c
11f1 : e8 c8 c0 Ød 90 f1 8e b0  43
11f9 : 02 a9 Ød 9d 5a 03 a9 00  2b
1201 : e8 9d 5a 03 20 cc ff a9  6b
1209 : 5a a0 03 20 cf 09 ae b0  da
1211 : 02 a9 2c 9d 5a 03 ac bb  8e
1219 : 02 b9 17 28 48 29 7f e8  60
1221 : 9d 5a 03 68 29 80 8d d0  28
1229 : 02 a9 2c e8 9d 5a 03 a9  34
1231 : 57 e8 9d 5a 03 e8 8e b1  c4
1239 : 02 a9 11 a0 2a 8d c6 02  96
1241 : 8c c7 02 a2 00 ac bb 02  de
1249 : b9 c7 27 8e ba 02 dd 67  83
1251 : 28 d0 03 4c 3d 13 e8 e0  fe
1259 : aa 90 ea 2c cf 02 30 1f  97
1261 : 20 d4 Øb 20 da Øb a9 74  48
1269 : a0 13 20 cf 09 ad af 02  56
1271 : 20 8d Øc 20 d5 Øe 20 72  92
1279 : Øa 20 d5 Øe 20 9b Øa ad  2d
1281 : ad 02 8d af 02 ad b1 02  e1
1289 : a2 5a a0 03 20 2a Øb 20  a1
1291 : e7 Øb aa 20 c9 ff ac bb  73
1299 : 02 be c7 27 20 b6 Ød a9  10
12a1 : 11 a0 2a 85 6f 84 70 ac  74
12a9 : bb 02 b9 17 28 29 7f c9  14
12b1 : 50 d0 Øa a9 00 20 d2 ff  6d
12b9 : a9 11 20 d2 ff a0 12 b1  fe
12c1 : 6b d0 3f 88 b1 6b d0 3a  a3
12c9 : 2c cf 02 30 06 20 cc ff  f8
12d1 : 20 d4 Øb ee bb 02 ce c1  86
12d9 : 02 d0 06 20 da Øe 4c 2d  73
12e1 : 08 2c cf 02 10 03 4c ca  13
12e9 : 11 a9 99 a0 13 20 cf 09  cd
12f1 : ad af 02 20 8d Øc 20 d5  60
12f9 : Øe 20 72 Øa 20 Øc Øb 4c  1c
1301 : c0 11 a0 00 b1 6f 2c d0  5b
1309 : 02 10 03 20 89 1c 8c b2  e9
1311 : 02 20 d2 ff ac b2 02 c8  d2
1319 : c0 80 90 e8 18 a5 6f 69  9a
1321 : 80 85 6f a5 70 69 00 85  52
1329 : 70 a0 11 38 b1 6b e9 01  55
1331 : 91 6b c8 b1 6b e9 00 91  09
1339 : 6b 4c be 12 a9 04 8d c0  2f
1341 : 02 ad ae 02 8d af 02 20  a4
1349 : fa 16 a2 31 ad c6 02 ac  90
1351 : c7 02 20 ee Øb ee c7 02  4b
1359 : ad c7 02 c9 cf 90 03 4c  ca
1361 : 50 Ød ce c0 02 f0 06 20  03
1369 : 15 18 4c 4b 13 ae ba 02  9c
1371 : 4c 57 12 Ød 12 20 c2 49  4f
1379 : 54 54 45 20 92 20 c3 c2  Øb
1381 : cd 2d c4 49 53 4b 45 54  8c
1389 : 54 45 20 49 4e 20 cc 41  4d
1391 : 55 46 57 45 52 4b 20 00  88
1399 : Ød 12 20 c2 49 54 54 45  23
13a1 : 20 92 20 c3 d0 2f cd 2d  a3
```

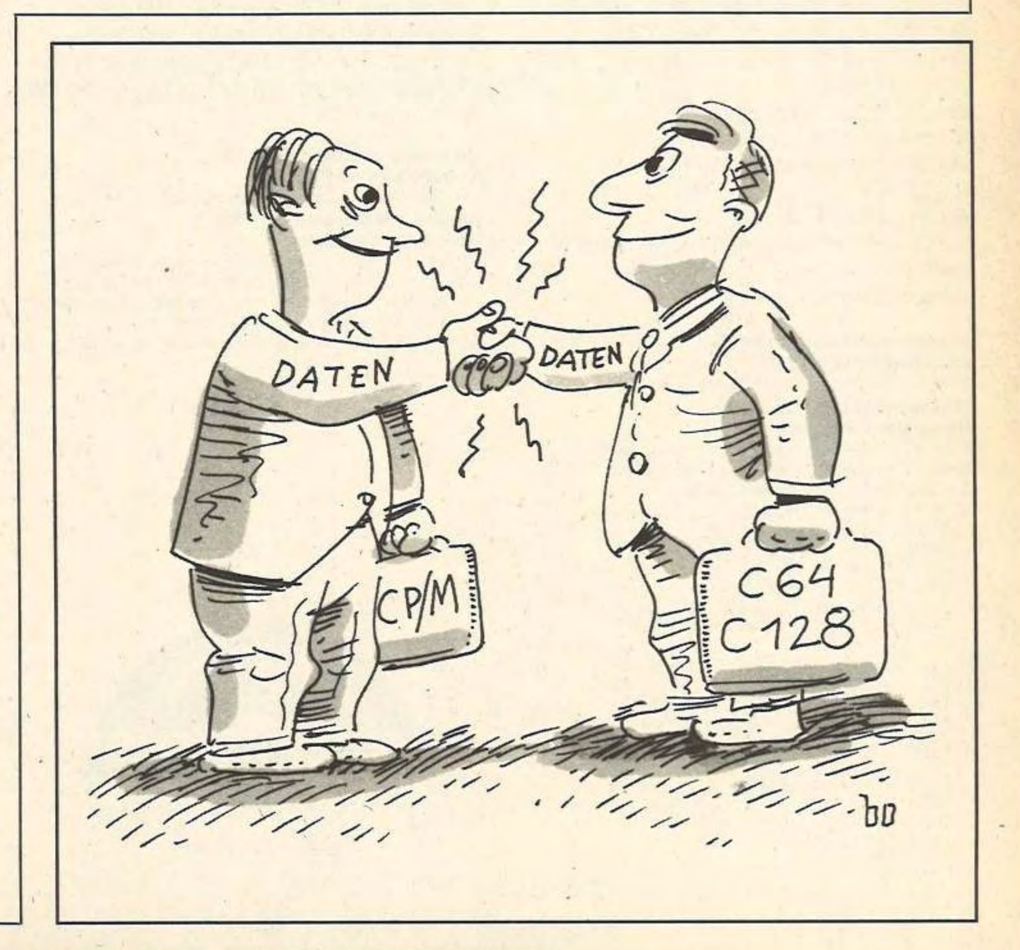
Listing 1. (Fortsetzung)

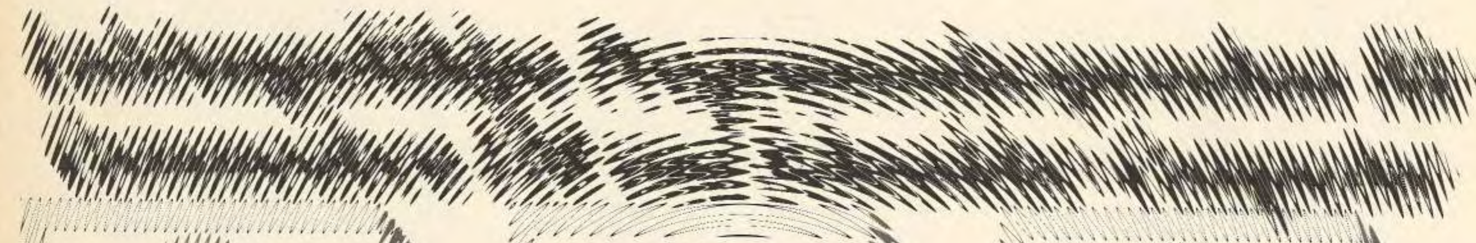
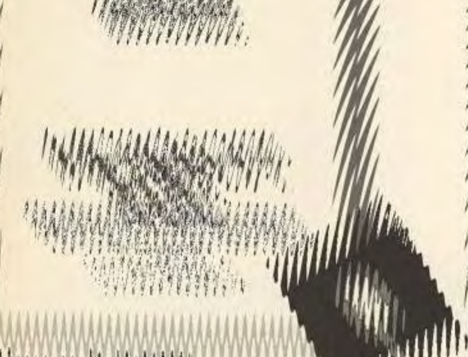
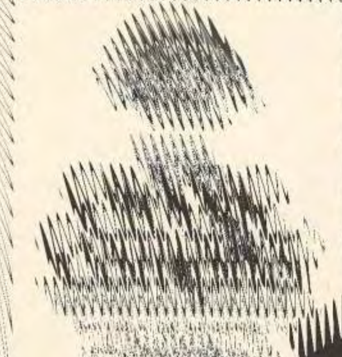
```
13a9 : c4 49 53 4b 45 54 54 45  23
13b1 : 20 49 4e 20 cc 41 55 46  c6
13b9 : 57 45 52 4b 20 00 0d 0d  01
13c1 : c5 53 20 57 49 52 44 20  9b
13c9 : 42 45 41 52 42 45 49 54  65
13d1 : 45 54 3a 0d 0d 00 0d 0d  90
13d9 : 0d 0d 20 20 20 20 20 20  3d
13e1 : 20 da 49 45 4c 2d cc 41  4d
13e9 : 55 46 57 45 52 4b 20 00  e0
13f1 : 20 90 09 a9 14 20 50 0e  70
13f9 : a9 00 8d c1 02 a0 50 99  d7
1401 : c6 27 99 16 28 88 d0 f7  7e
1409 : a2 01 20 8a 16 20 ba 0e  ef
1411 : a9 38 a0 14 20 cf 09 20  66
1419 : 79 0a c9 4a f0 4c c9 4e  88
1421 : d0 f5 a9 41 a0 14 20 cf  49
1429 : 09 2c cd 02 30 03 20 d5  43
1431 : 0e 20 57 0e 4c fa 14 4a  68
1439 : 41 20 2f 4e 45 49 4e 00  f8
1441 : 9d 9d 9d 9d 9d 9d 9d 9d  40
1449 : 20 20 12 20 4e 45 49 4e  d3
1451 : 20 92 20 20 00 9d 9d 9d  65
1459 : 9d 9d 9d 9d 9d 20 20 12  5f
1461 : 20 20 4a 41 20 20 92 20  da
1469 : 20 00 a9 56 a0 14 20 cf  89
1471 : 09 ad ba 02 38 e9 01 ac  70
1479 : c1 02 99 c7 27 2c cd 02  aa
1481 : 30 07 a9 59 a0 15 4c 8e  cb
1489 : 14 a9 6f a0 15 20 cf 09  05
1491 : 20 57 0e 20 79 0a c9 4a  88
1499 : f0 1d c9 4e d0 f5 2c cd  5d
14a1 : 02 30 07 a9 97 a0 15 4c  1e
14a9 : af 14 a9 b2 a0 15 20 cf  f6
14b1 : 09 20 57 0e 4c c0 14 ac  d6
14b9 : c1 02 a9 80 99 17 28 2c  41
14c1 : cd 02 30 40 a9 cd a0 15  59
14c9 : 20 cf 09 20 79 0a c9 0d  40
14d1 : f0 61 c9 50 f0 5d c9 53  b6
14d9 : f0 68 c9 55 d0 ed 20 52  bc
14e1 : 15 a9 85 a0 15 20 cf 09  e4
14e9 : a5 55 ac c1 02 19 17 28  32
14f1 : 99 17 28 ee c1 02 20 57  59
14f9 : 0e ae ba 02 ec bf 02 f0  04
1501 : 09 4c 0b 14 20 d5 0e 4c  f7
1509 : f4 14 a9 ec a0 15 20 cf  e2
1511 : 09 20 e6 09 20 79 0a c9  8f
1519 : 0d f0 0b c9 20 f0 07 c9  d4
1521 : 14 d0 f1 4c f1 13 ad c1  95
1529 : 02 f0 01 60 a9 12 a0 16  ca
1531 : 4c eb 0a 20 52 15 a9 8b  85
1539 : a0 15 20 cf 09 a5 50 4c  fd
1541 : eb 14 20 52 15 a9 91 a0  af
1549 : 15 20 cf 09 a5 53 4c eb  81
1551 : 14 a9 dd a0 15 4c cf 09  ca
1559 : 20 20 20 20 20 20 20 c1  9d
1561 : d3 c3 c9 c9 20 2d 3e 20  66
1569 : c3 c2 cd 20 20 00 20 20  c8
1571 : 20 20 20 20 20 c3 c2 cd  74
1579 : 20 20 20 2d 3e 20 c1 d3  eb
1581 : c3 c9 c9 00 12 55 53 52  59
1589 : 0d 00 12 50 52 47 0d 00  b9
1591 : 12 53 45 51 0d 00 9d 9d  4b
1599 : 9d 9d 9d c1 d3 c3 c9 c9  ba
15a1 : 20 20 20 20 20 20 20 20  a1
15a9 : 9d 9d 9d 9d 9d 9d 9d 9d  a8
15b1 : 00 9d 9d 9d 9d 9d c3 c2  f6
15b9 : cd 20 20 20 20 20 20 20  66
15c1 : 20 20 20 9d 9d 9d 9d 9d  26
15c9 : 9d 9d 9d 00 20 20 12 50  88
15d1 : 52 47 92 20 53 45 51 20  54
15d9 : 55 53 52 00 14 14 14 14  c7
15e1 : 14 14 14 14 14 14 14 14  e1
15e9 : 20 20 00 0d 0d 20 20 20  4e
15f1 : 20 c5 49 4e 47 41 42 45  22
15f9 : 20 52 49 43 48 54 49 47  d8
1601 : 20 3f 20 28 12 d3 d0 c1  54
1609 : c3 c5 92 2f c4 c5 cc 29  39
1611 : 00 0d 0d 12 20 3f 20 cb  32
1619 : 45 49 4e 20 c6 49 4c 45  0d
1621 : 20 47 45 57 41 45 48 4c  19
1629 : 54 20 0d 00 ae bf 02 ca  57
1631 : 8e bd 02 f0 4f ae bd 02  a2
1639 : 20 b6 0d a0 01 ae b0 02  58
1641 : b1 6b c9 a0 d0 02 a9 20  32
1649 : 48 bd 11 29 29 7f 9d 11  00
1651 : 29 68 dd 11 29 d0 28 e8  d3
1659 : c8 c0 09 90 e3 c8 b1 6b  f8
1661 : c9 a0 d0 02 a9 20 48 bd  27
1669 : 11 29 29 7f 9d 11 29 68  21
1671 : dd 11 29 d0 0a e8 c8 c0  c8
1679 : 0d 90 e3 ad bd 02 60 ce  88
1681 : bd 02 d0 b1 ad bf 02 18  bb
1689 : 60 8e ba 02 20 b6 0d 20  4b
1691 : cb 0d a0 01 8c b2 02 b1  f5
1699 : 6b 20 d2 ff ee b2 02 ac  af
16a1 : b2 02 c0 11 d0 f1 20 bd  3f
16a9 : 0e 2c cd 02 10 37 a0 00  be
16b1 : b1 6b 48 48 30 03 a9 2a  49
16b9 : 2c a9 20 20 d2 ff 68 29  e7
16c1 : 07 a8 8c b2 02 b9 eb 16  60
16c9 : 20 d2 ff ac b2 02 b9 f0  ec
16d1 : 16 20 d2 ff ac b2 02 b9  88
16d9 : f5 16 20 d2 ff 68 29 40  a4
16e1 : f0 03 a9 3c 2c a9 20 4c  6e
16e9 : d2 ff 44 53 50 55 52 45  ba
16f1 : 45 52 53 45 4c 51 47 52  ee
16f9 : 4c 2c ce 02 10 25 4c 7b  a2
1701 : 17 a9 09 a0 17 4c eb 0a  db
1709 : 0d 0d 12 20 3f 20 c6 41  b8
1711 : 4c 53 43 48 45 20 cb 2d  c0
1719 : c5 49 4e 48 45 49 54 20  50
1721 : 0d 0d 00 e0 88 b0 da 8e  68
1729 : c8 02 a9 00 8d c9 02 0e  a8
1731 : c8 02 2e c9 02 0e c8 02  77
1739 : 2e c9 02 a2 00 38 e8 ad  e1
1741 : c8 02 e9 11 8d c8 02 ad  29
1749 : c9 02 e9 00 8d c9 02 b0  1e
1751 : ed ca ad c8 02 69 11 8d  f3
1759 : c8 02 ad c9 02 69 00 8d  4d
1761 : c9 02 e0 0f 90 01 e8 e8  cc
1769 : e8 e8 8e a7 02 8e a9 02  9d
1771 : ad c8 02 8d a8 02 8d aa  db
1779 : 02 60 e0 aa b0 83 8e c8  2c
1781 : 02 a2 00 8e c9 02 0e c8  1d
1789 : 02 2e c9 02 0e c8 02 2e  e1
1791 : c9 02 a9 02 18 6d c8 02  1a
1799 : 8d c8 02 8d ab 02 a9 00  2e
17a1 : 6d c9 02 8d c9 02 8d ac  61
17a9 : 02 ca 38 e8 e0 23 b0 2b  7c
17b1 : e0 11 d0 08 ee c8 02 d0  2e
17b9 : 03 ee c9 02 38 ad c8 02  fe
17c1 : fd f2 17 8d c8 02 ad c9  96
17c9 : 02 e9 00 8d c9 02 b0 db  99
17d1 : ad c8 02 7d f2 17 8d c8  c2
17d9 : 02 f0 13 a8 a9 00 18 69  fb
17e1 : 05 dd f2 17 90 05 f0 03  6f
17e9 : fd f2 17 88 d0 f0 e8 d0  10
17f1 : 50 15 15 15 15 15 15 15  2c
17f9 : 15 15 15 15 15 15 15 15  f9
1801 : 15 15 13 13 13 13 13 13  04
1809 : 13 12 12 12 12 12 12 11  08
1811 : 11 11 11 11 2c ce [illegible] 30  b3
1819 : 2f ee aa 02 ad aa 02 ae  40
1821 : a9 02 c9 11 90 1b a9 00  e8
1829 : 8d aa 02 ee a9 02 ae a9  22
1831 : 02 e0 12 90 01 e8 e0 24  5d
1839 : 90 07 a9 5c a0 18 4c eb  17
1841 : 0a 8d a8 02 8e a7 02 60  6b
1849 : ad ab 02 8d c8 02 ad ac  ab
1851 : 02 8d c9 02 a9 01 a2 00  fa
1859 : 4c 95 17 0d 0d 12 20 3f  38
1861 : 20 46 41 4c 53 43 48 45  79
1869 : 52 20 d4 52 41 43 4b 20  e6
1871 : 0d 00 20 da 0e a9 ff 8d  2b
1879 : cd 02 20 50 0e 20 cf 0c  93
1881 : 20 a0 1c 20 90 09 a0 00  d0
1889 : 8c c2 02 8c c3 02 a9 d7  2b
1891 : a0 13 20 cf 09 20 73 0e  38
1899 : 8d ae 02 a0 ff 8c cf 02  ba
18a1 : cd ad 02 d0 03 ee cf 02  ca
18a9 : 2c cf 02 10 16 a9 99 a0  96
18b1 : 13 20 cf 09 ad ae 02 8d  5d
18b9 : af 02 20 8d 0c 20 72 0a  c3
18c1 : 20 9b 0a 20 90 09 a9 bf  ad
18c9 : a0 13 20 cf 09 ac c2 02  fa
18d1 : b9 17 28 29 80 8d d0 02  01
18d9 : be c7 27 20 b6 0d a2 00  a7
18e1 : a0 01 b1 6b c9 a0 f0 09  53
18e9 : 9d 5a 03 e8 c8 c0 11 90  89
18f1 : f1 8e b1 02 a9 0d 9d 5a  04
18f9 : 03 a9 00 e8 9d 5a 03 a9  fa
1901 : 5a a0 03 20 cf 09 ae b1  d4
1909 : 02 a9 2c 9d 5a 03 a0 00  df
1911 : b1 6b 29 07 a8 b9 eb 16  d7
1919 : e8 9d 5a 03 a9 2c e8 9d  a2
1921 : 5a 03 a9 52 e8 9d 5a 03  9c
1929 : a9 11 a0 2a 85 3f 84 40  ad
1931 : 85 41 84 42 ad ad 02 8d  2c
1939 : af 02 8a a2 5a a0 03 20  d7
1941 : 2a 0b ad ad 02 20 e7 0b  e9
1949 : aa 20 c6 ff a0 00 b1 6b  5c
1951 : c9 82 d0 06 20 cf ff 20  11
1959 : cf ff 20 cf ff a0 00 2c  87
1961 : d0 02 10 03 20 89 1c 91  79
1969 : 3f 20 29 0a 20 b7 ff d0  a5
1971 : 12 e6 3f d0 e5 e6 40 a5  c2
1979 : 40 c9 cf 90 dd 20 da 0e  0a
1981 : 4c 50 0d 20 cc ff 20 d4  34
1989 : 0b 2c cf 02 30 1c 20 da  f9
1991 : 0b a9 99 a0 13 20 cf 09  6f
1999 : ad ae 02 20 8d 0c 20 d5  87
19a1 : 0e 20 72 0a 20 d5 0e 20  c6
19a9 : 9b 0a ad ae 02 8d af 02  da
19b1 : 20 0c 0b 20 67 0f 20 a0  4f
19b9 : 0f a2 88 2c ce 02 10 02  02
19c1 : a2 aa a0 00 bd 66 28 d0  32
19c9 : 01 c8 ca d0 f7 8c b3 02  b2
19d1 : a5 40 38 e9 2a 4a 4a cd  9b
19d9 : b3 02 90 5c 48 a9 0e a0  88
19e1 : 1a 20 cf 09 68 a2 00 20  fc
19e9 : d3 0d a9 2c a0 1a 20 cf  2e
19f1 : 09 ad b3 02 a2 00 20 d3  50
19f9 : 0d a9 cb 20 d2 ff 20 d5  2b
1a01 : 0e 20 da 0e 20 72 0a 20  96
1a09 : e6 09 4c 2d 08 0d 0d 12  6e
1a11 : 20 3f 20 c4 49 53 4b 20  0e
1a19 : 56 4f 4c 4c 20 21 0d 0d  0d
1a21 : 42 45 4e 4f 45 54 49 47  2e
1a29 : 54 3a 00 cb 0d 56 4f 52  79
1a31 : 48 41 4e 44 45 4e 3a 00  e6
1a39 : a9 02 8d bd 02 18 a5 3f  f4
1a41 : e5 41 a5 40 e5 42 90 31  4d
1a49 : ae bd 02 bc 67 28 f0 03  6f
1a51 : e8 d0 f8 8e bd 02 88 98  f0
1a59 : 9d 67 28 20 fa 16 a9 04  c7
1a61 : 8d c0 02 a5 41 a4 42 a2  0b
1a69 : 32 20 ee 0b e6 42 ce c0  06
1a71 : 02 f0 ca 20 15 18 4c 64  ae
1a79 : 1a 38 a5 3f e9 11 8d c8  ef
1a81 : 02 a5 40 e9 2a 8d c9 02  dd
1a89 : a9 00 8d c4 02 8d c5 02  d6
1a91 : ee c4 02 d0 03 ee c5 02  3f
1a99 : 38 ad c8 02 e9 80 8d c8  85
1aa1 : 02 ad c9 02 e9 00 8d c9  95
1aa9 : 02 b0 e5 a2 00 20 11 1c  4e
1ab1 : a2 00 bd 5a 03 c9 2e d0  e7
1ab9 : 03 4c e8 1b 99 11 29 c8  d8
1ac1 : e8 ec b1 02 b0 11 e0 08  f3
1ac9 : 90 e8 bd 5a 03 c9 2e f0  a1
1ad1 : e8 e8 ec b1 02 90 f3 ad  6e
1ad9 : c5 02 18 69 0a a8 a2 0b  59
1ae1 : b9 11 29 9d 59 03 88 ca  86
1ae9 : d0 f6 ad c5 02 18 69 0e  fb
1af1 : 8d c5 02 a2 02 8e bd 02  c5
1af9 : a9 88 2c ce 02 10 02 a9  c7
1b01 : aa 8d c6 02 a9 10 8d bc  2e
1b09 : 02 bd 67 28 c9 ff f0 64  f2
1b11 : e8 ec c6 02 90 f3 ac c5  48
1b19 : 02 c8 a9 00 99 11 29 c8  42
1b21 : ce bc 02 d0 f7 98 38 e9  e1
1b29 : 11 a8 ad c4 02 99 11 29  16
1b31 : a2 02 a9 00 88 99 11 29  2b
1b39 : ca d0 f9 88 ad c3 02 99  2f
1b41 : 11 29 8e c3 02 a9 11 a0  f6
1b49 : 29 a2 32 20 ee 0b 20 d4  c5
1b51 : 0b 20 67 20 ee c2 02 ce  f5
1b59 : c1 02 f0 12 2c cf 02 10  03
1b61 : 03 4c ce 18 20 da 0b a9  99
1b69 : 74 a0 13 4c b2 18 20 da  9e
1b71 : 0e 4c 2d 08 ee c5 02 ac  70
1b79 : c5 02 8a 99 11 29 ce bc  24
1b81 : 02 d0 8d 8e bd 02 98 38  df
1b89 : e9 10 a8 38 ad c4 02 e9  88
1b91 : 80 8d c4 02 ad c5 02 e9  2e
1b99 : 00 8d c5 02 a9 80 99 11  39
1ba1 : 29 a9 00 a2 02 88 99 11  e0
1ba9 : 29 ca d0 f9 88 ad c3 02  b4
1bb1 : 99 11 29 ee c3 02 a9 11  10
1bb9 : a0 29 a2 32 20 ee 0b a2  c8
1bc1 : 10 8e bc 02 ae be 02 20  b1
1bc9 : 11 1c a2 00 bd 5a 03 99  7f
1bd1 : 11 29 c8 e8 e0 0c 90 f4  61
1bd9 : ad c5 02 18 69 0e 8d c5  b5
1be1 : 02 ae bd 02 4c 11 1b e8  76
1be9 : ec b1 02 90 03 4c d8 1a  6b
1bf1 : a9 03 8d c6 02 ad c5 02  01
1bf9 : 18 69 08 a8 bd 5a 03 99  cb
1c01 : 11 29 ce c6 02 f0 e6 c8  08
1c09 : e8 ec b1 02 90 ee b0 dd  13
1c11 : 8e be 02 a9 04 8d c0 02  68
1c19 : 20 fa 16 a9 11 a0 29 a2  71
1c21 : 31 20 ee 0b a0 00 b9 11  92
1c29 : 29 c9 e5 f0 1f 98 18 69  b8
1c31 : 20 a8 90 f2 ce c0 02 f0  05
1c39 : 06 20 15 18 4c 1c 1c ee  8c
1c41 : be 02 ae be 02 e0 02 90  d4
1c49 : ca 4c 67 1c a9 00 99 11  ba
1c51 : 29 c8 8c c5 02 98 18 69  d2
1c59 : 0b a8 a9 20 99 10 29 88  f7
1c61 : cc c5 02 d0 f7 60 20 da  63
```

Listing 1. (Fortsetzung)

```
1c69 : 0e a9 71 a0 1c 4c eb 0a  a4
1c71 : 0d 0d 12 20 3f 20 c4 49  28
1c79 : 52 45 43 54 4f 52 59 20  f6
1c81 : 56 4f 4c 4c 20 0d 0d 00  ba
1c89 : c9 41 90 06 c9 5b b0 03  18
1c91 : 69 20 60 c9 61 90 fb c9  7a
1c99 : 7b b0 f7 38 e9 20 60 20  d3
1ca1 : 90 09 a9 bc a0 1c 20 cf  c3
1ca9 : 09 20 79 0a c9 32 f0 4c  ec
1cb1 : c9 0d f0 48 c9 31 d0 f1  93
1cb9 : 4c f1 13 0d 0d 20 20 20  f7
1cc1 : 20 20 2d 20 20 31 20 20  8d
1cc9 : 2d 20 20 20 20 c5 49 4e  04
1cd1 : 5a 45 4c 41 55 53 57 41  d9
1cd9 : 48 4c 0d 0d 0d 20 20 20  bf
1ce1 : 20 20 2d 20 12 20 32 20  8c
1ce9 : 92 20 2d 20 20 20 20 c2  e4
1cf1 : 4c 4f 43 4b 41 55 53 57  da
1cf9 : 41 48 4c 00 20 90 09 a9  6f
1d01 : 7f a0 1f 20 cf 09 a9 00  88
1d09 : 8d bd 02 8d be 02 a8 a9  99
1d11 : ff 99 11 2a c8 d0 fa 98  97
1d19 : a0 50 99 c6 27 99 16 28  08
1d21 : 88 d0 f7 a0 0b a2 04 18  29
1d29 : 20 f0 ff a9 01 8d bb 02  66
1d31 : 8d ba 02 38 20 f0 ff 8c  46
1d39 : b3 02 a2 04 a0 00 18 20  c1
1d41 : f0 ff 20 35 1f ac b3 02  0a
1d49 : a2 12 18 20 f0 ff 20 79  81
1d51 : 0a c9 0d f0 64 c9 11 f0  5c
1d59 : da c9 91 d0 0c ad ba 02  b3
1d61 : 38 e9 14 b0 cb a9 01 d0  59
1d69 : c7 c9 20 f0 e1 c9 14 d0  99
1d71 : 0e 48 ce be 02 10 24 a9  b3
1d79 : 00 8d be 02 68 f0 cf c9  11
1d81 : 3a b0 29 c9 2c f0 08 c9  95
1d89 : 2d f0 04 c9 30 90 1d ac  be
1d91 : be 02 48 29 0f 99 11 2a  de
1d99 : ee be 02 a9 14 20 d2 ff  2a
1da1 : 68 20 d2 ff a9 5f 20 d2  8a
1da9 : ff 4c 4f 1d 20 e6 09 4c  3c
1db1 : 4f 1d a9 c9 a0 1f 4c eb  3e
1db9 : 0a a0 00 8c bb 02 8c c1  26
1dc1 : 02 8c cb 02 8c c7 02 a0  8d
1dc9 : 00 8c c8 02 8c c9 02 8c  ba
1dd1 : ca 02 a2 00 ac bb 02 b9  69
1dd9 : 11 2a c9 ff f0 15 c9 0c  69
1de1 : b0 11 c8 e8 e0 03 d0 ef  b2
1de9 : b9 11 2a c9 ff f0 04 c9  1a
1df1 : 0c 90 bf e0 00 f0 bb ac  21
1df9 : bb 02 b9 11 2a ee bb 02  53
1e01 : ca 9d c8 02 d0 f1 a9 00  4f
1e09 : a0 02 be c8 02 f0 09 18  6f
1e11 : 79 0a 20 b0 9d ca d0 f8  13
1e19 : 88 10 ef 8d c6 02 ac bb  fd
1e21 : 02 b9 11 2a c9 0d d0 03  d8
1e29 : 4c 59 1e 2c cb 02 10 03  42
1e31 : 4c 79 1e ad c6 02 ac c1  2a
1e39 : 02 cd bf 02 90 03 4c b3  0c
1e41 : 1d 99 c7 27 8d c7 02 ee  ff
1e49 : c1 02 ac bb 02 ee bb 02  38
1e51 : cc be 02 f0 52 4c c8 1d  00
1e59 : ad c6 02 8d c7 02 ee bb  5b
1e61 : 02 a9 ff 8d cb 02 ac bb  e0
1e69 : 02 cc be 02 f0 03 4c c8  ab
1e71 : 1d ae bf 02 ca 8e c6 02  56
1e79 : a9 00 8d cb 02 ad c6 02  ac
1e81 : cd c7 02 b0 03 4c b3 1d  64
1e89 : d0 03 4c 37 1e ee c6 02  4d
1e91 : ad c7 02 ac c1 02 99 c7  5a
1e99 : 27 ee c1 02 18 69 01 cd  54
1ea1 : c6 02 d0 ef 4c 4b 1e 20  72
1ea9 : d5 0e 20 d5 0e 2c cd 02  c6
1eb1 : 30 07 a9 59 a0 15 4c be  5c
1eb9 : 1e a9 6f a0 15 20 cf 09  3f
1ec1 : a9 0d a0 20 20 cf 09 20  02
1ec9 : 79 0a c9 4a f0 14 c9 0d  f4
1ed1 : f0 04 c9 4e d0 f1 2c cd  e8
1ed9 : 02 30 16 a9 97 a0 15 4c  19
1ee1 : f6 1e a9 80 ac c1 02 99  75
1ee9 : 17 28 88 c0 ff d0 f8 f0  9b
1ef1 : 07 a9 b2 a0 15 20 cf 09  31
1ef9 : 2c cd 02 10 01 60 20 d5  ce
1f01 : 0e 20 d5 0e a9 cd a0 15  0c
1f09 : 20 cf 09 20 79 0a c9 0d  80
1f11 : f0 0d c9 50 f0 0b c9 53  39
1f19 : f0 07 c9 55 d0 ed 2c a9  2a
1f21 : 50 ac c1 02 48 68 48 19  93
1f29 : 17 28 99 17 28 88 c0 ff  67
1f31 : d0 f3 68 60 ae bf 02 ca  a7
1f39 : ec ba 02 b0 01 60 ae bf  66
1f41 : 02 ca ec ba 02 90 1e ae  b5
1f49 : ba 02 20 8a 16 20 d5 0e  34
1f51 : ee bd 02 a9 0a cd bd 02  de
1f59 : d0 e4 a9 00 8d bd 02 20  15
1f61 : d5 0e 4c d5 0e a9 0a cd  fd
1f69 : bd 02 f0 ee a9 e3 a0 1f  bc
1f71 : 20 cf 09 20 d5 0e ee bd  c4
1f79 : 02 ee ba 02 d0 e7 13 11  9c
1f81 : 11 11 11 11 11 11 11 11  81
1f89 : 11 11 11 11 11 11 11 12  8b
1f91 : 43 52 44 92 20 3d 3e 20  86
1f99 : d3 45 49 54 45 20 57 45  29
1fa1 : 49 54 45 52 2c 12 43 52  b5
1fa9 : 55 92 20 3d 3e 20 d3 45  b6
1fb1 : 49 54 45 20 5a 55 52 55  be
1fb9 : 45 43 4b 0d c1 55 53 57  d7
1fc1 : 41 48 4c 20 3a 20 5f 00  60
1fc9 : 0d 0d 12 20 3f 20 c6 41  78
1fd1 : 4c 53 43 48 45 20 c5 49  a0
1fd9 : 4e 47 41 42 45 20 21 0d  57
1fe1 : 0d 00 20 20 20 20 20 20  be
1fe9 : 20 20 20 20 20 20 20 20  e9
1ff1 : 20 20 20 20 20 20 20 20  f1
1ff9 : 20 20 20 20 20 20 20 20  f9
2001 : 20 20 20 20 20 20 20 20  01
2009 : 00 01 0a 64 20 28 4a 2f  64
2011 : 12 4e 92 29 20 3f 9d 9d  c2
2019 : 9d 9d 9d 9d 9d 9d 00 48  f7
2021 : 8e b0 02 8c b2 02 a0 00  d7
2029 : a9 52 e0 32 d0 02 a9 57  ec
2031 : 8d e2 07 ad a7 02 20 4e  4e
2039 : 20 c8 a9 20 99 e2 07 ad  54
2041 : a8 02 20 4e 20 68 ae b0  1e
2049 : 02 ac b2 02 60 a2 00 38  1a
2051 : e9 0a 90 03 e8 d0 f9 69  94
2059 : 0a 48 8a 20 60 20 68 c8  68
2061 : 09 30 99 e2 07 60 48 8c  f3
2069 : b2 02 a0 06 a9 20 99 e1  cb
2071 : 07 88 10 fa ac b2 02 68  59
2079 : 60 20 90 09 a9 cc a0 20  f2
2081 : 20 cf 09 20 f1 20 20 73  57
2089 : 0e 8d af 02 8d ae 02 20  21
2091 : ea 0f a2 00 8e ba 02 a9  c6
2099 : e5 9d 11 29 e8 d0 fa a9  0b
20a1 : 04 8d c0 02 20 fa 16 a2  54
20a9 : 32 a9 11 a0 29 20 ee 0b  6e
20b1 : ce c0 02 f0 06 20 15 18  64
20b9 : 4c a8 20 ee ba 02 ae ba  2b
20c1 : 02 e0 02 90 da 20 da 0e  fc
20c9 : 4c 2d 08 0d 0d c3 d0 2f  e0
20d1 : cd 2d c4 49 52 45 43 54  94
20d9 : 4f 52 59 20 4c 4f 45 53  a7
20e1 : 43 48 45 4e 20 3f 20 28  30
20e9 : 12 4a 92 2f 4e 29 20 00  5a
20f1 : 20 79 0a c9 0d f0 0e c9  ae
20f9 : 4a f0 0a c9 4e d0 f1 20  eb
2101 : e6 09 4c 2d 08 a9 4a 4c  b4
2109 : d2 ff a9 00 2c a9 ff 8d  70
2111 : ce 02 20 90 09 a9 8a a0  44
2119 : 21 20 cf 09 20 f1 20 20  b2
2121 : 73 0e 8d af 02 8d ae 02  40
2129 : 20 e0 0b ae af 02 a0 0f  fe
2131 : 20 ba ff a9 0f a2 7b a0  19
2139 : 21 20 bd ff 20 c0 ff 90  03
2141 : 03 4c bb 0a 20 89 0b 20  55
2149 : 0c 0b 2c ce 02 30 03 4c  06
2151 : 93 20 a2 00 bd b3 21 9d  d6
2159 : 11 29 e8 e0 24 90 f5 a9  47
2161 : 00 9d 11 29 e8 d0 fa 8e  b7
2169 : a8 02 e8 8e a7 02 a2 32  98
2171 : a9 11 a0 29 20 ee 0b 4c  2e
2179 : 93 20 4e 3a 43 50 2f 4d  05
2181 : 20 44 49 53 4b 2c 5a 43  86
2189 : 0d 0d 0d c4 49 53 4b 45  e0
2191 : 54 54 45 20 28 c3 d0 2f  a7
2199 : cd 29 20 46 4f 52 4d 41  0b
21a1 : 54 49 45 52 45 4e 20 3f  fb
21a9 : 20 28 12 4a 92 2f 4e 29  d9
21b1 : 20 00 43 42 4d 00 00 00  bf
21b9 : 00 00 00 78 20 84 ff a9  42
21c1 : 3e 8d 00 ff a9 c3 8d ee  93
21c9 : ff a9 08 8d ef ff a9 00  f6
21d1 : 8d f0 ff 4c d0 ff ee 07  36
```

Listing 1. (Ende). »CP/M <-> CBM« kopiert Dateien vom C64 nach CP/M 2.2 oder 3.0 und umgekehrt, formatiert CP/M 2.2- oder 3.0-Disketten (einseitig) und stellt somit die Schnittstelle zwischen CP/M und den Commodore-eigenen Aufzeichnungsformaten her. Bitte verwenden Sie zur Eingabe den MSE (Eingabehinweise auf den Seiten 93 – 95).





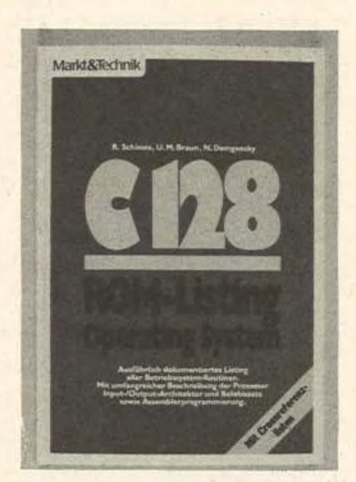
C-64/SX-64 Computer Handbuch

Ein Buch, das dem Einsteiger eine gewaltige Fülle an Informationen liefert, aber auch dem Fortgeschrittenen noch eine Menge bietet ist das C-64/SX-64 Computer Handbuch von Raeto West. Das neunseitige Inhaltsverzeichnis zeigt schon, daß praktisch alle Gebiete rund um den C 64 behandelt werden, wobei die Informationen dennoch nicht erschlagend wirken. Fast alle Themen sind mit einer Menge kleinerer Programme unterlegt, die das Geschriebene plastisch darstellen und vertiefen. Bilden die ersten drei Kapitel mit Beschreibung der Tastaturfunktionen, Syntax und Befehlssatz etc. ein Grundgerüst als Ergänzung des Commodore-Handbuches, so wird in den folgenden 14 Kapiteln Wissen zur Verfügung gestellt, das bei weitem über das Handbuch hinausreicht. Wie man effektiv in Basic programmiert, zum Beispiel sucht, sortiert und mischt, erfährt man im vierten Kapitel auch anhand zahlreicher kurzer Programme. Eine detaillierte Beschreibung der Hardware findet sich in Kapitel 5, unter anderem beispielsweise alle Ein- und Ausgänge des C 64/SX 64. Für die dann schon Fortgeschrittenen wird es in Kapitel 6 besonders interessant. Hier wird genau beschrieben, wie und wo Basic-Programme abgelegt sind und was es mit dem Aufräumen von nicht mehr benötigten Zeichenketten (Garbage Collection) auf sich hat. Die Kapitel 7 bis 11 behandeln auf über 100 Seiten den Befehlssatz der 6502/6510 CPU und stellen typische Verfahren der Maschinenprogrammierung sowie deren Einsatz vor. Auskunft über wichtige Speicheradressen im RAM und ROM liefert Kapitel 11 und erweitert damit den Anwendungsbereich des zuvor Dargestellten enorm. Die folgenden Kapitel behandeln Grafik, Ton und Musik, Kassettenrecorder und Disketten als Speichermedien, sowie Drukker, Plotter und Modems. Abgerundet wird dieses Buch durch einen ausgiebigen Anhang, der nicht nur durch Tabellen glänzt, sondern auch Programme wie einen Maschinensprachemonitor, Schnelladeprogramme für Kassette und Diskette, sowie eine Programmeingabehilfe enthält. Einziger kleiner Schönheitsfehler dieses Buches ist, daß bei der Übersetzung aus dem Englischen die Namen der Programme nicht ebenfalls angepaßt wurden. Da dies das einzige Relikt darstellt und alle Texte in den Programmen, wie auch deren Dokumentation, selbstverständlich deutschsprachig sind, ist dieser Schönheitsfehler nicht weiter von Belang.

Dieses Werk kann jedem C64/SX 64-Besitzer wärmstens empfohlen werden und leistet Ihnen auch als Nachschlagewerk in den nächsten Jahren wertvolle Hilfe.

(R. Sauer/bj)

Info: Raeto West, C-64/SX-64 Computer Handbuch, te-wi Verlag GmbH, etwa 500 Seiten, ISBN 3-921803-24-1, Preis: 66 Mark



C 128-ROM-Listing

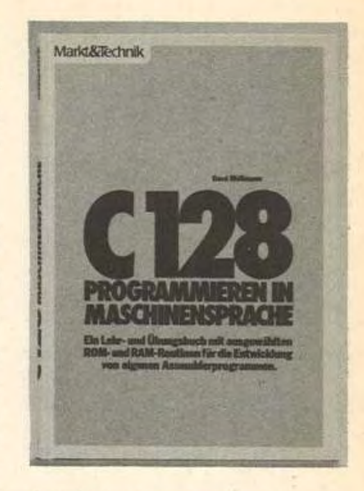
Ein unentbehrliches Arbeitsmittel für Maschinensprache-programmierer ist das C128-ROM-Listing. Gab es für den C128 bislang nur eine Dokumentation von Monitor und Betriebssystem, so liegt nun ein vollständiges ROM-Listing vor, das die gesamten 44 KByte-ROM, also auch den 28 KByte langen Basic-7.0-Interpreter, umfaßt. Das Buch »C128-ROM-Listing« aus der Commodore-Sachbuchreihe, die vom Markt&Technik Verlag vertrieben wird, erklärt zunächst die komplizierte Speicherverwaltung des C128. Dieses Kapitel ist zwar nur 20 Seiten stark, aber sehr informativ.

Auf etwa 25 Seiten werden dann noch die Videocontroller VIC und VDC, der Soundchip SID, die CIAs in Kurzform behandelt und eine, im Gegensatz zum C128-Handbuch, vollständige Aufstellung der Systemadressen geboten. Leider wurden dabei die Adressen, die dem Anwender zur Verfügung stehen (also vom System nicht verwendet werden), nicht erwähnt. Den Hauptteil macht mit 375 Seiten das ROM-Listing selbst aus, welches sowohl den Basic 7.0-Interpreter, als auch das Betriebssystem dem Assemblerprogrammierer offenlegt. Das ROM-Listing ist, um es vollständig in einem einzigen Buch unterzubringen, etwas kleiner als der normale Text gedruckt, aber dennoch sehr gut lesbar. Erfreulich ist, daß die Kommentare in vernünftigem Maß gegeben werden: Befehle, die für sich selbst sprechen, werden nicht dokumentiert, was die Übersichtlichkeit enorm erhöht und dadurch das Nachschlagen unterstützt. Die äußere Struktur des ROM-Listings überzeugt durch Klarheit und Übersichtlichkeit, da es sich um einen Quelltext-Ausdruck handelt und die eingesetzten Label eine schnelle Suche von Sprungzielen ermöglichen. Die Labelnamen bestehen aus der hexadezimalen Adreßangabe mit einem vorangestellten »h«, also zum Beispiel »hFFD2« für $FFD2; dadurch kann man die tatsächliche Adresse so einfach wie bei einem Monitor-Ausdruck ausfindig machen, was bei der Analyse von ROM-Routinen sehr hilfreich ist.

Fazit: Das vorliegende Buch ist für jeden C128-Maschinenprogrammierer ein unbedingtes »Muß«. Aufgrund des immensen Umfangs (456 Seiten), des großen Nutzens und eines guten Schlagwortregisters ist der Preis von 58 Mark angemessen. Man hätte lediglich noch eine Einführung in die Arbeit mit ROM-Listings erwartet, der geübte Leser wird darauf aber sicher verzichten können.

(Florian Müller/bj)

Info: Commodore-Sachbuchreihe, Dr. Ruprecht, C128-ROM-Listing, Markt&Technik Verlag, 456 Seiten, ISBN 3-89090-212-X, Preis: 58 Mark



C 128: Programmieren in Maschinensprache

Das im Markt&Technik Verlag erschienene Buch »C128: Programmieren in Maschinensprache« von Gerd Möllmann ist kein Lehrbuch für 6502/6510-Assembler-Programmierung, sondern vielmehr eine Weiterführung und vertiefende Literatur, die dem 6502-Vertrauten den Weg zur effektiven Assembler-Programmierung mit dem C128 erleichtert und auch als Nachschlagewerk wertvolle Dienste leistet.

Im ersten Kapitel werden die neuen Bausteine des C128 (MMU, VDC) behandelt, aber auch die Register der schon bekannten ICs wie VIC II, SID und CIAs kommen hier nicht zu kurz. Das zweite Kapitel beschreibt die Routinen der »Common Area«, also der Kernel- und Interpreter-Unterprogramme, im gemeinsamen RAM-Bereich von $0000 bis $03FF. Im dritten Kapitel wird auf 50 Seiten das Betriebssystem analysiert, daran angehängt die Kernel-Vektoren. Im fünften und längsten Kapitel wird der Basic-Interpreter sorgfältig zerlegt. Das sechste Kapitel beschäftigt sich mit den dazugehörigen Basic-Vektoren in der erweiterten Zero-Page und das siebte Kapitel beschreibt deren Einsatzmöglichkeiten.

Der Autor hat mit diesem Buch ein ausführliches Nachschlagewerk für die Assembler-Programmierung des C128 geschaffen, das so gut wie möglich mit Beispielprogrammen unterlegt ist und trotz der Informationsvielfalt immer noch übersichtlich bleibt.

(Jörg Sahlmann/bj)

Info: Gerd Möllmann, C128: Programmieren in Maschinensprache, Markt & Technik Verlag, 270 Seiten, ISBN 3-89090-213-8, Preis: 52 Mark



Erfolgreicher mit dem VC 64 arbeiten

Mit diesem Buch wird der Autor all diejenigen C64-Besitzer erreichen, die sich gerade überlegen, wie sie den Sprung »ins Eingemachte«, nämlich die Programmierung in Maschinensprache, schaffen sollen. Dieses Werk berücksichtigt nun die ganze Komplexität des Themenbereiches und widmet sich diesem »von der Pike« auf.

Der Verfasser, selbst Diplomingenieur, beginnt das Buch mit der Vermittlung von Grundlagen der Digitaltechnik, beschreibt die Funktion verschiedener Zahlensysteme und gibt als Abschluß des ersten Teils des Buches einen Überblick über das grundlegende Funktionsprinzip eines Mikrocomputers, um dann speziell auf den Prozessor 6502 und seinen Befehlssatz einzugehen.

Der zweite Teil des Buches beschäftigt sich dann im besonderen mit dem C64 und seinen Eigenheiten. In gesonderten Kapiteln wird der Speicheraufbau und die Speicherverwaltung dem Leser genauestens dargestellt, im folgenden Abschnitt wird das Augenmerk besonders auf die Interface-Bausteine des C64 gerichtet. Dabei werden die einzelnen Register und deren Programmierung sehr genau beschrieben.

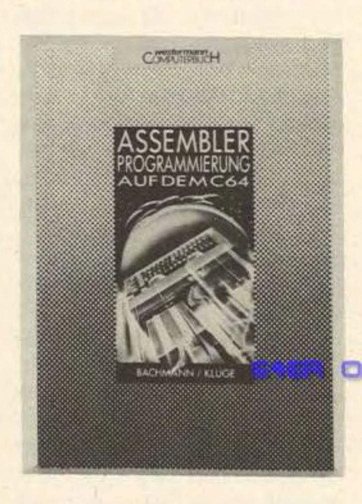
Im dritten Teil werden einige kleine, aber nützliche Programme besprochen, die den Anwender auch zu selbständigen Arbeiten anregen sollen. Dieser Teil des Buches ist gerade für den Einsteiger sehr empfehlenswert, hat er hier doch die Möglichkeit, seine bereits vorhandenen Kenntnisse durch Übung zu erweitern.

Weiterhin findet man über das ganze Buch verstreut zahllose Tabellen, die sowohl eine Hilfe für den Anfänger, als auch ein Nachschlagewerk für den Routinier darstellen. Im Anhang gibt es dann noch eine ausführliche Liste der ROM-Routinen mit hexadezimalen und dezimalen Adressen, sowie eine Umrechnungstabelle von hexadezimal nach dezimal und umgekehrt.

Alles in allem kann man feststellen: Sowohl für den Einsteiger als auch für den geübten Basic-Programmierer ist dieses Buch sehr gut geeignet, um sich alle nötigen Grundlagen im Umgang mit der Maschinensprache anzueignen.

(Udo Reetz/ev/bj)

Info: Franz Wunderlich, Erfolgreicher mit dem VC 64 arbeiten, Franzis-Verlag GmbH, 192 Seiten, ISBN 3-7723-7781-5, Preis: 38 Mark



Assemblerprogrammierung auf dem C64

Daß dieses Werk aus einem Schulbuchverlag (Westermann) stammt, erkennt man sofort am didaktischen Aufbau und den Übungsaufgaben am Ende jedes Abschnittes, deren Nutzen aber durch einen Lösungsteil erheblich vergrößert werden könnte. Im ersten Kapitel werden die nötigen mathematischen und schaltungstechnischen Kenntnisse vermittelt, wobei aber anzumerken ist, daß man als Nicht-Elektroniker nach den ersten Seiten mittlere bis größere Verständnisschwierigkeiten haben wird. Neben den üblichen Erklärungen des Befehlssatzes und der Adressierungsarten ist besonders positiv anzumerken, daß auch solche Themen, vor denen sich andere Autoren drücken, klar und ausführlich behandelt werden: Anwendung von Monitor- und Assembler-Programm, Sound-Programmierung, Einbindung von Maschinenroutinen in Basic-Programme und die Interrupt-Technik. Viel zu kurz kommen mathematische Anwendungen, denn die Arbeit mit Fließkommazahlen wird überhaupt nicht erklärt, während die äußerst unkomfortable BCD-Darstellung bis ins letzte Detail zerlegt wird. Nur sehr wenige ROM-Routinen werden erklärt, was aber allgemein ein Problem in vergleichbarer Literatur ist, wenn man von wenigen Ausnahmen absieht. Die Stärke des Buches gegenüber den Mitbewerbern liegt zweifelsohne darin, daß man ohne Umschweife erfährt, wie man Assembler-Programme eingibt, im Speicher unterbringt, ablaufen läßt und weiterverwendet. Dies mag dem Assembler-Unerfahrenen so trivial wie in Basic erscheinen, stellt aber in der Praxis ein Problem für sich dar. Erfreulich ist, daß auch eine Diskette mit Assembler- und Monitor-Programm bezogen werden kann, welches wirklich leistungsfähig ist, wenn die Diskette auch mit 79 Mark leicht überteuert scheint. Wer bereits einen anderen Prozessor als einen 65xx (C64) programmiert hat (zum Beispiel den Z80 des Sinclair ZX81) oder über Elektronik-Kenntnisse verfügt, ist mit diesem preiswerten Buch bestens bedient. Andernfalls fällt die Lektüre nicht ganz so leicht.

(Florian Müller/bj)

Info: Walter Bachmann/ Norbert Kluge, Assemblerprogrammierung auf dem C64, Westermann Schulbuchverlag, 270 Seiten, ISBN 3-14-138813-X, Preis: 29,80 Mark; Diskette mit Monitor und Assembler: Bestellnummer 138013, Preis: 79 Mark



Pascal mit dem C64

Das erste, das sofort angenehm an diesem Buch auffällt, ist die mitgelieferte Programmdiskette, auf der sich ein komplettes und lauffähiges Pascal-System, inklusive Full-Screen-Editor und Compiler, befindet. Anders betrachtet könnte man es natürlich auch als Programmpaket mit einer sehr, sehr ausführlichen Anleitung bezeichnen. Im ersten Kapitel wird der Leser dem Thema »Pascal« durch allgemeine Einleitungen der Art »Warum Pascal?« und »Was macht ein Compiler?« nähergebracht. Danach folgt eine schrittweise Einführung in die Bedienung des Systems anhand von Beispielen. Der zweite und größte Teil des Buches stellt eine vollständige »Einführung in Pascal« dar. Besonders ausführlich werden die komplizierteren Datentypen (Array, Record und File), sowie die verschiedenen Datenstrukturen (dynamische, lineare, Listen, Bäume, etc.) behandelt. Dieses Kapitel wirkt sehr ausgewogen, da wirklich alle Elemente der Sprache vorgestellt werden. Außerdem werden im Text einige sehr nützliche Unterprogramme und Algorithmen entwickelt (zum Beispiel Quicksort). Abgerundet wird dieser Abschnitt durch die vielen kleinen Aufgabenstellungen am Ende jedes Teilkapitels, die den Leser zum selbständigen Arbeiten und Lernen gekonnt animieren. In Kapitel drei schließlich geht es um Tips und Tricks speziell für den Commodore 64. Hier wird gezeigt, wie man auf das Betriebssystem und die Floppystation zugreift. Der vierte Teil behandelt ausschließlich die Beschreibung des mitgelieferten Pascal-Systems. Dabei wird sowohl auf die Bedienung der einzelnen Programme, als auch auf die Besonderheiten des Compilers eingegangen. Dem Gesamteindruck nach, eignet sich dieses Buch in erster Linie für Einsteiger in die Programmiersprache Pascal, bietet aber auch dem erfahrenen Pascal-Benutzer sicherlich einige wertvolle Anregungen. Der interessierte Pascal-Anwender erhält sowohl die künftige Arbeitsgrundlage zum Erstellen, Testen und Betreiben von Pascal-Programmen, als auch eine Einführung in diese Sprache mit Anregungen für weitere eigene Programmentwicklungen. »Pascal mit dem C64« kann allen C64-Besitzern empfohlen werden, die die Sprache »Pascal« erlernen möchten.

(Christoph Bergmann/bj)

Info: Florian Matthes, Pascal mit dem C64, Markt&Technik Verlag, 215 Seiten, ISBN-3-89090-222-7, Preis: 52 Mark mit Programmdiskette (Pascal-System)

Checksummer V3 und MSE

Diese beiden Programme sind unentbehrlich beim Abtippen unserer Listings. Sie helfen Tippfehler zu vermeiden und sparen eine Menge Zeit.

Nobody is perfect. Jeder Computer-Fan, egal ob blutiger Anfänger oder ausgefuchster Profi, macht beim Abtippen von Programmen Tippfehler. Diese Fehler später zu finden, kann ein langwieriges Unterfangen werden.

Deshalb haben wir für Sie die Programme »Checksummer V3« und »MSE« (MaschinenSpracheEditor) entwickelt. Der Checksummer ist für Basic-Programme und der MSE für Maschinensprache-Listings zuständig.

Der Checksummer

Zuerst einmal müssen Sie das Checksummer-Programm (siehe Listing 1) abtippen. Dabei sollten Sie äußerst sorgfältig vorgehen, vor allem bei den Zahlen in den DATA-Zeilen 20 bis 30. Wenn Sie trotzdem noch einen Tippfehler gemacht haben, meldet sich das Programm später mit einem entsprechenden Hinweis. Wenn Sie fertig sind, müssen Sie das Programm auf Diskette oder Kassette speichern.

Jetzt geht es los:

1. Starten Sie den Checksummer durch die Eingabe von »RUN« und dem Drücken der <RETURN>-Taste.
2. Wenn die Meldung »Checksummer aktiviert...« auf dem Bildschirm erscheint, haben Sie keinen Tippfehler gemacht und der Checksummer ist nun eingeschaltet.
3. Zum Löschen des Basic-Programms geben Sie bitte »NEW« ein. Keine Angst, der Checksummer selbst wird dadurch nicht gelöscht.
4. Nun können wir den Checksummer testen. Geben Sie bitte folgende Zeile ein und drücken Sie die <RETURN>-Taste:

1 REM

In der linken oberen Bildschirmecke sehen Sie nun die Prüfsumme über der eben eingegebenen Basic-Zeile. Sie muß <63> lauten. Dem Checksummer ist es übrigens egal, ob Sie »1 REM« oder »1REM« eintippen. Nur innerhalb von Anführungszeichen ist die richtige Anzahl an Leerzeichen wichtig. Diese Prüfsummen erscheinen (sofern Sie den Checksummer eingeschaltet haben) immer dann, wenn Sie eine Basic-Zeile eintippen und dann die <RETURN>-Taste drücken. In der 64'er finden Sie die Prüfsummen immer am Ende jeder Programmzeile.

Diese Zahlen dürfen Sie NICHT mit abtippen. Sie dienen lediglich zur Kontrolle, ob Sie alles richtig eingegeben haben.

Als Beispiel können Sie sich Bild 1 betrachten. Am rechten Rand jeder Spalte sehen Sie die Prüfsummen in eckigen Klammern.

Damit sind wir beim zweiten wichtigen Punkt: Sehen Sie sich die Zeile 341 von Listing 2 genauer an. Nach dem ersten Anführungszeichen nach dem PRINT-Befehl sehen Sie ein Zeichen, das Sie auf der Tastatur des C 64 vergeblich suchen werden: die geschweifte Klammer { }. Immer, wenn Sie in einem unserer Listings diese Klammern sehen, dürfen Sie das, was innerhalb der Klammern steht, nicht eintippen. Sie müssen die entsprechende Taste drücken. Beispiel:

10 PRINT "{CLR}"

bedeutet: Nach dem Anführungszeichen die »Bildschirm-löschen«-Taste drücken (<SHIFT+CLR/HOME>). In Tabelle 1 sehen Sie eine Zusammenfassung aller möglichen Steuertasten und dem entsprechenden Klartext.

Weiterhin sehen Sie in Listing 2 (MSE) in Zeile 341 ein unterstrichenes »O« nach dem »P«. Das bedeutet, daß Sie ein »O« zusammen mit der <SHIFT>-Taste drücken müssen, also <SHIFT+O>. Wenn ein Zeichen »überstrichen« ist, müssen Sie dieses zusammen mit der <CBM>-Taste eingeben. Die <CBM>-Taste befindet sich ganz links unten auf der Tastatur und hat die Aufschrift »C=«. Auf dem Bildschirm sehen Sie die entsprechenden Grafikzeichen (siehe Handbuch, Seite 133).

Der MSE

Der MSE dient zur Eingabe von Maschinensprache-Programmen. Als erstes müssen Sie den sogenannten »MSE-Lader« (Listing 2) abtippen. Dieser erzeugt erst das eigentliche MSE-Programm auf Diskette oder Kassette.

Wichtig: Vor dem Eintippen des MSE-Laders müssen Sie unbedingt ein paar Befehle eingeben (ohne Basic-Zeilennummer): POKE 44,32 : POKE 8192,0 : NEW

Jetzt können Sie beginnen, das Listing 2 abzutippen. Der MSE-Lader erkennt zwar, wenn Sie beim Eintippen der DATA-Zeilen einen Fehler gemacht haben, aber wenn Sie ganz sicher gehen möchten, sollten Sie den Checksummer vor dem Eintippen aktivieren. Die Prüfsummen für den MSE-Lader finden Sie am Ende der jeweiligen Programmzeilen.

Wenn Sie das Listing 2 nicht auf einmal abtippen möchten, müssen Sie vor jedem neuen Laden des Programms unbedingt die oben genannte POKE-Zeile eingeben!

Datasetten-Besitzer müssen die »8« am Ende von Zeile 343 in eine »1« ändern.

CTRL steht für Control-Taste, so bedeutet {CTRL+A}, daß Sie die Control-Taste und die Taste »A« drücken müssen. Im folgenden steht:

Code	Bedeutung
{DOWN}	Taste neben rechtem Shift, Cursor unten
{UP}	Shift-Taste & Taste neben rechtem Shift; Cursor hoch
{CLR}	Shift-Taste & 2. Taste ganz rechts oben
{INST}	Shift-Taste & Taste ganz rechts oben
{HOME}	2. Taste von ganz rechts oben
{DEL}	Taste ganz rechts oben
{RIGHT}	Taste ganz rechts unten
{LEFT}	Shift-Taste & Taste unten rechts
{SPACE}	Leertaste
{SHFIT-Space}	Shift-Taste & Leertaste
{F1} bis {F8}	Funktionstasten
{RETURN}	Shift-Taste & Return
{BLACK}	Control-Taste & 1
{WHITE}	Control-Taste & 2
{RED}	Control-Taste & 3
{CYAN}	Control-Taste & 4
{PURPLE}	Control-Taste & 5
{GREEN}	Control-Taste & 6
{BLUE}	Control-Taste & 7
{YELLOW}	Control-Taste & 8
{RVSON}	Control-Taste & 9
{RVOFF}	Control-Taste & 0
{ORANGE}	Commodore-Taste & 1
{BROWN}	Commodore-Taste & 2
{LIG.RED}	Commodore-Taste & 3
{GREY 1}	Commodore-Taste & 4
{GREY 2}	Commodore-Taste & 5
{LIG.GREEN}	Commodore-Taste & 6
{LIG.BLUE}	Commodore-Taste & 7
{GREY 3}	Commodore-Taste & 8

Tabelle 1. Die Steuerbefehle in den Listings

Wenn Sie alles richtig gemacht haben und das Programm fehlerfrei abgetippt wurde, speichert es sich selbst auf Diskette oder Kassette unter dem Namen »MSE V1.0«. Dieses fertige MSE-Programm laden Sie dann bei Bedarf wie ein normales Basic-Programm und starten es mit »RUN«.

So arbeitet man mit dem MSE

Als erstes möchte der MSE den Namen des zu bearbeitenden Programms wissen. Dieser steht in der ersten Zeile unserer MSE-Listings. Dann müssen Sie die Start- und Endadresse des Programms eingeben. Dies sind die letzten beiden, vierstelligen Hexadezimalzahlen in der ersten Zeile unserer Listings.

Wenn Sie ein Programm von Diskette oder Kassette laden wollen, um an einer bestimmten Stelle weiterzutippen oder noch eine Korrektur vorzunehmen, geben Sie auf die Frage nach der Startadresse ein »L« ein. Danach müssen Sie <D> oder <T> drücken, je nachdem, ob Sie von Diskette oder Kassette (»tape«) laden möchten. Wenn das Programm unter diesem Namen nicht auf der Diskette vorhanden ist, oder ein sonstiger Ladefehler vorlag, meldet sich der MSE mit »I/O-ERROR«. In so einem Fall drücken Sie <RUN/STOP+RESTORE> und geben einfach noch einmal »RUN« ein.

Beim Abtippen geben Sie nach und nach die abgedruckten Buchstaben und Zahlen des jeweiligen Listings ohne die Freiräume dazwischen ein. Wenn Sie in einer Zeile einen Tippfehler gemacht haben, meldet sich der MSE sofort mit einem Brummton und der Meldung »EINGABEFEHLER«. Nach einem Druck auf die <RETURN>-Taste können Sie mit der <DEL>-Taste den Fehler korrigieren.

Wenn Sie das gewünschte Programm vollständig eingegeben haben, speichert es der MSE automatisch auf Diskette oder Kassette.

Bei längeren Listings ist es unwahrscheinlich, daß Sie das komplette Programm auf einmal eingeben. Sie können Ihre bisherige Tipparbeit jederzeit durch <CTRL+S> auf Diskette oder Kassette speichern und Ihr Werk später fortsetzen. Sie sollten sich dann allerdings im Heft markieren, wie weit Sie beim Abtippen gekommen sind! Später geben Sie dann nach dem Laden des ersten Programmteils <CTRL+N> ein und auf die dann folgende Frage nach der Startadresse die Zeilennummer (Adresse), bei der Sie aufgehört haben zu tippen.

<CTRL+M> erlaubt Ihnen jederzeit, Ihr Werk listen zu lassen. Durch <SPACE> können Sie weiterlisten lassen und durch <RUN/STOP> das Listen abbrechen.

Wenn Sie einen Drucker besitzen, können Sie das Programm auch mit <CTRL+P> ausdrucken.

Mit <CTRL+L> wird das Programm noch einmal neu in Ihren C 64 geladen.

(F. Lonczewski/N. Mann/D. Weineck/tr)

```
5 PRINT CHR$(14)                                    <242>
10 PRINT"{CLR}"                                     <254>
20 PRINT"Q*****************************W"           <130>
30 PRINT"{4DOWN,2SPACE}TEST{SPACE,BLUE,6SP
   ACE}"                                            <022>
40 PRINT"BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"            <108>
```

Bild 1. Die Bedeutung der Steuerzeichen wird im Text erklärt

Bild 1. In Zeile 10 müssen Sie nach den Anführungsstrichen die <SHIFT+CLR/HOME>-Taste drücken und nicht die Klammern mit dem Wort <CLR>. In Zeile 20 drücken Sie nach den Anführungsstrichen die Commodore-Taste und den Buchstaben <Q>, gefolgt von mehreren <SHIFT>- und Stern-Tasten und zum Schluß die Commodore-Taste und den Buchstaben <W>. In Zeile 30 ist es viermal die <CRSR>-unten-Taste, gefolgt von zweimaliger Leertaste, dann <SHIFT+T> und normal EST, zum Schluß noch einmal die Leertaste, die Farbtaste Blau <CTRL+7> und sechsmal die Leertaste. Zeile 40 besteht lediglich aus mehreren Grafikzeichen, die mit der Commodore-Taste und <B> erzeugt werden.

```
10 PRINT"CHECKSUMMER FUER C 64"
11 PRINT:PRINT"EINEN MOMENT, BITTE ..."
12 FOR I=828 TO 864:READ A:POKE I,A:PS=PS+
   A:NEXT I
13 IF PS<>5765 THEN PRINT"TIPPFEHLER IN DE
   N ZEILEN 20 BIS 22":END
14 SYS 828:PS=0:FOR I=58464 TO 58583:READ
   A:POKE I,A:PS=PS+A:NEXT I
15 IF PS<>16147 THEN PRINT"TIPPFEHLER IN D
   EN ZEILEN 22 BIS 30":END
16 POKE 1,53:POKE 42289,96:POKE 42290,228
17 PRINT"CHECKSUMMER AKTIVIERT."
18 PRINT:PRINT" AUSSCHALTEN : POKE1,55 ODE
   R"SPC(27)"<RUN/STOP+RESTORE>"
19 PRINT:PRINT" ANSCHALTEN : POKE1,53"
20 DATA 169,0,133,254,162,1,189,93,3,133,2
   55,160,0,177,254
21 DATA 145,254,136,208,249,230,255,165,25
   5,221,95,3,208,238,202
22 DATA 16,230,96,160,224,192,0,160,2,169,
   0,170,133,254,177
23 DATA 95,240,40,201,32,208,3,200,208,245
   ,133,255,138,41,7
24 DATA 170,240,14,72,165,255,24,42,105,0,
   202,208,249,133,255
25 DATA 104,170,232,165,255,24,101,254,133
   ,254,76,111,228,192,4
26 DATA 48,219,198,214,165,214,72,162,3,16
   9,32,157,1,4,189
27 DATA 212,228,32,210,255,208,12,0,92,72,
   32,201,255,170,104
28 DATA 144,1,138,96,202,16,228,166,254,16
   9,0,32,205,189,169
29 DATA 62,32,210,255,104,133,214,32,108,2
   29,169,141,32,210,255
30 DATA 76,128,164,9,60,18,19
```

© 64'er

Listing 1. Der »Checksummer 64 V3« für Basic-Listings

```
100 REM ****************************         <091>
110 REM *                          *         <159>
120 REM *       M S E   LADER      *         <206>
130 REM *                          *         <179>
220 REM ****************************         <211>
230 REM                                      <036>
240 DIM H(75): FOR I=0 TO 9                  <113>
250 H(48+I)=I: H(65+I)=I+10:NEXT             <041>
260 FOR I=2048 TO 3755 : READ A$             <198>
270 H=ASC(LEFT$(A$,1)):L=ASC(RIGHT$(A$,1))   <199>
280 D=H(H)*16+H(L):S=S+D:POKE I,D            <219>
290 A=A+1:IF A<20 THEN NEXT:A=-1             <141>
300 PRINT " ZEILE:";1000+Z;                  <011>
310 READ V :Z=Z+1:IF V=S THEN 330            <218>
320 PRINT"PRUEFSUMMENFEHLER !":STOP          <138>
330 IF A<0 THEN 341                          <221>
340 S=0:A=0:PRINT:NEXT                       <046>
341 PRINT"{CLR}P043,1:P044,8:P045,172:P046
    ,14                                      <010>
342 POKE 631,19:POKE 632,13:POKE 633,13:PO
    KE 198,3                                 <249>
343 PRINT"{3DOWN}SAVE"CHR$(34)"MSE V1.0"CH
    R$(34)",8                                <171>
344 END                                      <092>
1000 DATA 00,0B,08,0A,00,9E,32,30,36,31,00
     ,00,00,A2,08,A9,36,85,A4,A9, 1247        <119>
1001 DATA 08,85,A5,A9,00,85,A6,A9,B0,85,A7
     ,A0,00,B1,A4,91,A6,C8,D0,F9, 2888        <054>
1002 DATA E6,A5,E6,A7,CA,D0,F2,A9,36,85,01
     ,4C,00,B0,20,D1,B1,A9,06,8D, 2787        <144>
1003 DATA 21,D0,A9,03,8D,20,D0,8D,86,02,A0
     ,B3,A9,74,20,FF,B1,A0,B3,A9, 2667        <237>
1004 DATA B9,20,FF,B1,A0,00,20,CF,FF,99,01
     ,02,C8,C9,0D,D0,F5,88,F0,D2, 2912        <217>
```

Listing 2. Der »MSE« zur Eingabe von Maschinensprache-Programmen

```
1005 DATA C0,0F,90,02,A0,0E,8C,00,02,20,EA
     ,B1,A0,B3,A9,CF,20,FF,B1,20, 2323    <013>
1006 DATA 8E,B4,85,FC,85,62,20,8E,B4,85,FB
     ,85,61,20,A7,B4,D0,20,A0,B3, 2864    <199>
1007 DATA A9,E5,20,FF,B1,20,8E,B4,85,60,20
     ,8E,B4,85,5F,20,A7,B4,D0,0A, 2624    <091>
1008 DATA A5,61,C5,5F,A5,62,E5,60,90,06,20
     ,43,B3,4C,3A,B0,A9,AA,A0,00, 2379    <167>
1009 DATA 91,FB,E6,FB,D0,02,E6,FC,20,3F,B2
     ,90,EF,4C,FB,B4,A2,02,86,58, 3118    <152>
1010 DATA A9,A6,A0,9D,20,F2,B1,20,E4,FF,F0
     ,FB,C9,30,90,0C,C9,47,B0,08, 2970    <231>
1011 DATA C9,3A,90,0B,C9,41,B0,07,C9,14,D0
     ,0F,4C,0B,B1,20,D2,FF,A6,58, 2322    <121>
1012 DATA 95,F7,C6,58,D0,D2,60,AE,8D,02,F0
     ,26,C9,0C,D0,03,4C,0B,B6,C9, 2685    <057>
1013 DATA 13,D0,03,4C,8B,B5,C9,0D,D0,03,4C
     ,BA,B4,C9,10,D0,03,4C,68,B5, 2282    <225>
1014 DATA C9,0E,D0,06,20,5F,B4,4C,64,B1,4C
     ,92,B0,A5,F9,20,02,B1,0A,0A, 2132    <208>
1015 DATA 0A,0A,85,F9,A5,F8,20,02,B1,05,F9
     ,60,C9,3A,90,02,69,08,29,0F, 1950    <092>
1016 DATA 60,A6,59,E0,08,90,1F,A6,58,E0,02
     ,B0,06,20,D2,FF,4C,8E,B0,C6, 2509    <188>
1017 DATA 59,A0,14,A9,92,20,F2,B1,CA,D0,FA
     ,84,57,68,68,4C,8B,B1,A6,D3, 2891    <197>
1018 DATA E0,08,B0,03,4C,92,B0,20,D2,FF,A6
     ,58,E0,02,90,09,C6,59,20,D2, 2468    <049>
1019 DATA FF,C6,58,D0,F9,4C,8E,B0,48,4A,4A
     ,4A,4A,20,59,B1,68,29,0F,C9, 2419    <035>
1020 DATA 0A,90,02,69,06,69,30,4C,D2,FF,A2
     ,FC,9A,20,D1,B1,20,48,B2,20, 2261    <073>
1021 DATA EA,B1,20,9F,B2,A5,FC,20,4E,B1,A5
     ,FB,20,4E,B1,20,ED,B1,A9,3A, 2860    <148>
1022 DATA A0,20,20,F2,B1,A9,00,85,59,20,8E
     ,B0,20,ED,B1,A4,59,20,EF,B0, 2530    <233>
1023 DATA 91,FB,C8,84,59,C0,08,90,EC,20,10
     ,B2,A9,12,20,D2,FF,20,8E,B0, 2657    <105>
1024 DATA 20,EF,B0,C5,FF,F0,0D,20,43,B3,A9
     ,14,A0,14,20,F2,B1,4C,A2,B1, 2665    <034>
1025 DATA A9,92,20,D2,FF,20,33,B2,20,E0,B2
     ,20,3F,B2,90,9F,4C,8B,B5,A9, 2648    <123>
1026 DATA 93,20,D2,FF,A2,00,A9,03,9D,00,D8
     ,9D,00,D9,9D,00,DA,9D,00,DB, 2476    <237>
1027 DATA E8,D0,EF,60,A9,0D,2C,A9,20,4C,D2
     ,FF,20,D2,FF,98,4C,D2,FF,20, 2965    <160>
1028 DATA E4,FF,F0,FB,60,84,5D,85,5C,A0,00
     ,B1,5C,F0,06,20,D2,FF,C8,D0, 3100    <077>
1029 DATA F6,60,A5,FB,85,5A,A0,00,84,5B,B1
     ,FB,18,65,5A,85,5A,90,02,E6, 2606    <156>
1030 DATA 5B,06,5A,26,5B,C8,C0,08,90,EC,A5
     ,5A,65,5B,85,FF,60,18,A5,FB, 2467    <219>
1031 DATA 69,08,85,FB,90,02,E6,FC,60,A5,FB
     ,C5,5F,A5,FC,E5,60,60,A0,B3, 3106    <183>
1032 DATA A9,FB,20,FF,B1,A0,01,B9,00,02,20
     ,D2,FF,CC,00,02,C8,90,F4,A9, 2692    <098>
1033 DATA 10,ED,00,02,AA,20,ED,B1,CA,D0,FA
     ,A5,62,20,4E,B1,A5,61,20,4E, 2453    <236>
1034 DATA B1,20,ED,B1,A5,60,20,4E,B1,A5,5F
     ,20,4E,B1,A9,9F,20,D2,FF,20, 2575    <038>
1035 DATA EA,B1,24,5E,10,01,60,A9,12,20,D2
     ,FF,A2,28,20,ED,B1,CA,D0,FA, 2646    <161>
1036 DATA A9,92,4C,D2,FF,A5,D6,C9,16,B0,01
     ,60,A9,A0,85,A4,A9,78,85,A6, 2945    <204>
1037 DATA A9,04,85,A5,85,A7,A2,13,A0,27,B1
     ,A4,91,A6,88,10,F9,CA,F0,19, 2671    <208>
1038 DATA 18,A5,A4,69,28,85,A4,90,02,E6,A5
     ,18,A5,A6,69,28,85,A6,90,E0, 2503    <251>
1039 DATA E6,A7,4C,B6,B2,A9,91,4C,D2,FF,A9
     ,0F,8D,18,D4,A9,00,8D,05,D4, 2776    <000>
1040 DATA A9,F7,8D,06,D4,A9,11,8D,04,D4,A9
     ,32,8D,01,D4,A9,00,8D,00,D4, 2413    <126>
1041 DATA A0,80,20,09,B3,A9,10,8D,04,D4,60
     ,A2,FF,CA,D0,FD,88,D0,F8,60, 2914    <240>
1042 DATA A9,0F,8D,18,D4,A9,2D,8D,05,D4,A9
     ,A5,8D,06,D4,A9,21,8D,04,D4, 2385    <119>
1043 DATA A9,07,8D,01,D4,A9,05,8D,00,D4,A0
     ,FF,20,09,B3,A9,20,8D,04,D4, 2250    <078>
1044 DATA A9,00,8D,01,D4,8D,00,D4,60,38,20
     ,F0,FF,8A,48,98,48,18,A0,06, 2179    <175>
1045 DATA A2,18,20,F0,FF,A0,B4,A9,0A,20,FF
     ,B1,20,12,B3,20,E4,FF,F0,FB, 2931    <093>
1046 DATA A2,1D,A9,14,20,D2,FF,CA,D0,FA,68
     ,A8,68,AA,18,4C,F0,FF,0D,0D, 2704    <088>
1047 DATA 0D,20,20,20,20,20,20,20,4D,41,53
     ,43,48,49,4E,45,4E,53,50,52, 1144    <216>
1048 DATA 41,43,48,45,20,2D,20,45,44,49,54
     ,4F,52,20,0D,0D,20,20,20,20, 1023    <038>
1049 DATA 20,20,20,20,56,4F,4E,20,4E,2E,4D
     ,41,4E,4E,20,26,20,44,2E,57, 1128    <206>
1050 DATA 45,49,4E,45,43,4B,00,0D,0D,0D,20
     ,20,20,50,52,4F,47,52,41,4D, 1102    <117>
1051 DATA 4D,4E,41,4D,45,20,3A,20,00,0D,0D
     ,20,20,20,53,54,41,52,54,41, 1073    <095>
1052 DATA 44,52,45,53,53,45,20,3A,20,24,00
     ,0D,0D,20,20,20,45,4E,44,41, 1014    <129>
1053 DATA 44,52,45,53,53,45,20,20,20,3A,20
     ,24,00,92,05,20,50,52,4F,47, 1171    <217>
1054 DATA 52,41,4D,4D,20,3A,20,00,12,20,20
     ,2A,2A,2A,20,46,41,4C,53,43, 1024    <027>
1055 DATA 48,45,20,45,49,4E,47,41,42,45,20
     ,2A,2A,2A,20,20,92,00,0D,0D, 1058    <098>
1056 DATA 2A,2A,2A,20,45,4E,44,45,20,2A,2A
     ,2A,00,13,05,20,20,12,44,92, 920     <148>
1057 DATA 49,53,4B,20,4F,44,45,52,20,12,54
     ,92,41,50,45,0D,00,13,20,20, 1151    <035>
1058 DATA 49,2F,4F,20,2D,20,46,45,48,4C,45
     ,52,00,20,D1,B1,20,48,B2,A0, 1606    <012>
1059 DATA B3,A9,CF,20,FF,B1,20,8E,B4,85,FC
     ,20,8E,B4,85,FB,C5,61,A5,FC, 3207    <251>
1060 DATA E5,62,90,23,A5,FB,C5,5F,A5,FC,E5
     ,60,B0,19,20,A7,B4,D0,14,60, 2860    <112>
1061 DATA 20,A7,B4,F0,0C,85,F9,20,A7,B4,F0
     ,05,85,F8,4C,EF,B0,68,68,20, 2749    <088>
1062 DATA 43,B3,4C,5F,B4,20,CF,FF,C9,4C,D0
     ,09,20,D1,B1,20,48,B2,4C,0B, 2372    <046>
1063 DATA B6,C9,0D,60,A9,00,85,5E,20,5F,B4
     ,20,EA,B1,20,0D,B5,24,5E,30, 2042    <120>
1064 DATA 05,20,E4,FF,F0,FB,20,E1,FF,F0,26
     ,20,9F,B2,24,5E,10,09,20,4E, 2435    <198>
1065 DATA B5,20,0D,B5,20,60,B5,20,33,B2,20
     ,3F,B2,90,D7,A0,B4,A9,28,20, 2190    <207>
1066 DATA FF,B1,20,E4,FF,C9,0D,D0,F9,A9,00
     ,85,5E,A5,61,85,FB,A5,62,85, 3056    <240>
1067 DATA FC,20,E0,B2,4C,64,B1,A5,FC,20,4E
     ,B1,A5,FB,85,FF,20,4E,B1,A9, 3003    <221>
1068 DATA 20,A0,3A,20,F2,B1,A0,00,20,ED,B1
     ,B1,FB,20,4E,B1,C8,C0,08,90, 2566    <070>
1069 DATA F3,20,ED,B1,24,5E,30,03,A9,12,2C
     ,A9,20,20,D2,FF,20,10,B2,A5, 2190    <059>
1070 DATA FF,20,4E,B1,A9,92,20,D2,FF,4C,EA
     ,B1,A9,FF,85,B8,85,B9,A9,04, 3073    <029>
1071 DATA 85,BA,20,C0,FF,A2,FF,4C,C9,FF,20
     ,CC,FF,A9,FF,4C,C3,FF,20,5F, 3315    <189>
1072 DATA B4,A9,80,85,5E,20,4E,B5,20,48,B2
     ,A2,24,A9,2D,20,D2,FF,CA,D0, 2596    <111>
1073 DATA FA,20,EA,B1,20,EA,B1,20,60,B5,4C
     ,C1,B4,20,B8,B5,A6,5F,A4,60, 2812    <015>
1074 DATA A9,61,20,D8,FF,B0,0A,20,B7,FF,29
     ,BF,D0,03,4C,FB,B4,A9,01,20, 2577    <201>
1075 DATA C3,FF,20,68,B6,A0,B4,A9,4F,20,FF
     ,B1,20,F9,B1,4C,FB,B4,20,68, 2921    <237>
1076 DATA B6,A9,37,A0,B4,20,FF,B1,20,F9,B1
     ,A2,08,C9,44,F0,06,A2,01,C9, 2717    <213>
1077 DATA 54,D0,F1,A9,01,A8,20,BA,FF,A0,00
     ,E0,01,F0,1A,A9,40,8D,20,02, 2403    <101>
1078 DATA A9,3A,8D,21,02,B9,01,02,99,22,02
     ,C8,CC,00,02,90,F4,C8,C8,D0, 2182    <127>
1079 DATA 0C,B9,01,02,99,20,02,C8,CC,00,02
     ,D0,F4,98,A2,20,A0,02,4C,BD, 2018    <025>
1080 DATA FF,20,B8,B5,A5,BA,C9,08,90,33,A6
     ,B9,86,57,A9,01,20,C3,FF,A9, 2800    <022>
1081 DATA 60,85,B9,20,C0,FF,B0,28,A5,BA,20
     ,B4,FF,A5,B9,20,96,FF,20,A5, 2911    <053>
1082 DATA FF,85,61,A5,90,4A,4A,B0,13,20,A5
     ,FF,85,62,20,AB,FF,A5,57,85, 2663    <214>
1083 DATA B9,A9,00,20,D5,FF,90,03,4C,A3,B5
     ,86,5F,84,60,A5,BA,C9,01,D0, 2639    <131>
1084 DATA 0A,AD,3D,03,85,61,AD,3E,03,85,62
     ,4C,FB,B4,A9,13,20,D2,FF,A2, 2300    <120>
1085 DATA 1C,20,ED,B1,CA,D0,FA,60, 1230    <214>
```


Listing 2. Der »MSE« zur Eingabe von Maschinensprache-Programmen (Schluß)

Schnelle Fill-Routine in Maschinensprache

Maschinensprache lernt man am besten an Hand von Beispielen. In Form eines dokumentierten Quellcodes erklären wir Ihnen die Fill-Routine aus dem Programm »HiRes-Master«, Sonderheft 11/86, bei dem es sich um eine der schnellsten Grafik-Erweiterungen für den C 64 handelt.

Gut dokumentierte Assemblerlistings erleichtern den Einstieg in die Maschinensprache erheblich. Was nutzt es dem Anfänger, wenn er zwar alle 6510-Maschinenbefehle kennt, aber nicht weiß, wie sie anzuwenden sind? Nichts! Erst durch Assemblerlistings wird das Zusammenspiel der einzelnen Maschinenbefehle klar und deutlich. Plötzlich versteht man, wie 16-Bit-Additionen oder -Subtraktionen funktionieren. Man lernt, in Maschinensprache mit Multiplikationen und Divisionen umzugehen und vieles mehr.

Bevor Sie sich jetzt mit dem Listing beschäftigen, schauen Sie sich zunächst die Flußdiagramme (Bild 1) zum Fill-Befehl an. Dadurch bekommen Sie einen Überblick über das Programm und verstehen, um was es überhaupt geht.

Die Flußdiagramme

Nachdem das Programm (Listing 1) mit einem Assembler in Maschinencode übersetzt beziehungsweise das MSE, Listing (Listing 2) abgetippt und gespeichert wurde, läßt es sich mit SYS 7 * 4096,x,y starten. Die Parameter »x« (0 bis 319) und »y« (0 bis 199) geben den Startpunkt an, ab dem die Fläche gefüllt werden soll. Falls der Koordinatenpunkt »x,y« schon gesetzt ist, gilt die Fläche als bereits gefüllt.

An einem Beispiel soll Ihnen der Algorithmus erklärt werden:

Nehmen wir an, wir haben auf dem Bildschirm eine horizontale Linie mit den Anfangs- und Endkoordinaten Xa=40, Ya=100 und Xe=280, Ye=100 (Bild 2). Nun rufen wir die Fill-Routine mit SYS 7 * 4096,150,50 auf. Zunächst überprüft das Programm, ob der Punkt an den Koordinaten X=150 und Y=50 gesetzt ist oder nicht. Ist er gesetzt, gilt die Fläche als bereits gefüllt und das Programm wird beendet. In unserem Fall ist der Punkt aber nicht gesetzt. Jetzt werden nacheinander folgende Schritte abgearbeitet. Dabei entspricht der Ausdruck »Plotkoordinate« dem Punkt, der gesetzt werden soll:

Schritt 1. Die Y-Koordinate wird solange erniedrigt, bis an der Plotkoordinate X,Y (in unserem Beispiel 150,Y) entweder ein Punkt gesetzt oder der obere Bildschirmrand erreicht ist (in unserem Fall wird der obere Bildschirmrand bei der Plotkoordinate X=150,Y=0 erreicht).

Schritt 2. Der Punkt mit der Plotkoordinate X,Y wird gesetzt.

Schritt 3. Die Koordinatenpunkte X-1,Y und X+1,Y (hier 149,0 und 151,0) werden überprüft. Ist einer der überprüften Punkte gesetzt, passiert nichts. Ist er nicht gesetzt, merkt sich das Programm diesen Punkt in zwei Feldern X, Y mit dem Feldzeiger »P«, der beim Start der Fill-Routine auf 0 gesetzt wurde <X(P)=X+/-1, Y(P)=Y>. (Da in unserem Fall weder der linke noch der rechte Punkt gesetzt ist, merkt sich das Programm beide (X(0)=149, Y(0)=0; X(1)=151, Y(1)=0).

Schritt 4. Die Fags »FL« oder »FR« werden dann auf 1 gesetzt, wenn in Schritt 3 ein getesteter, linker oder rechter Punkt ins Feld eingetragen wurde. <Wird FL auf 1 gesetzt, darf sich das Programm keine weiteren linken Punkte merken. Ist FL=0, ist das Merken linker Punkte erlaubt. Das gleiche gilt bei FR für die rechten Punkte.

Schritt 5. Ohne sich linke oder rechte Punkte zu merken, werden entlang der aktuellen Y-Achse (bei X=150) solange Punkte untereinander gesetzt, bis ein Punkt an der Plotkoordinate gesetzt ist (X=150, Y=100).

Schritt 6. Die Plotkoordinaten werden neu gesetzt: X=X(1), Y=Y(1) (151, 0) und der Feldzeiger P wird um 1 erniedrigt. Außerdem setzt das Programm beide Flags wieder auf 0 und läßt damit das Beschreiben des Feldes wieder zu.

(Anmerkung: Ab diesem Durchlauf wird immer nur der Punkt X+1,0 ins Feld übernommen, weil der Punkt X-1 schon gesetzt ist.)

Eine Fläche wird gefüllt

Die Schritte 1 bis 6 wiederholen sich solange, bis die rechte Fläche (ab der Anfangskoordinaten X=150) über der horizontalen Linie gefüllt ist. Dabei sind alle Ausdrücke in runden Klammern zu überlesen, denn sie sollten Ihnen nur die ersten Programmschritte näher erläutern.

Ist die besagte Fläche ausgefüllt, steht in der Variablen X der Wert 281 und in Y der Wert 0. Im Feld sind zwei Koordinaten gespeichert (X(0)=149,Y(0)=0; X(1)=282,Y(1)=0) und der Feldzeiger steht auf 1.

Der Punkt an der Plotkoordinate wird gesetzt.

Im nächsten Durchlauf (X=281,Y=1) setzt das Programm FR auf 1 (Merken rechter Punkte ist untersagt) und FL bleibt 0 (Merken linker Punkte ist erlaubt, werden aber nicht ins Feld übernommen, weil linke Punkte schon gesetzt sind).

Es werden jetzt an der Y-Achse bei X=281 solange Punkte untereinander gesetzt, bis bei X-1 ein freier Punkt gefunden wird (X-1=280, Y=101). Diese Koordinate wird ins Feld übernommen, der Feldzeiger wird um 1 erhöht und das Merken linker Punkte wird untersagt.

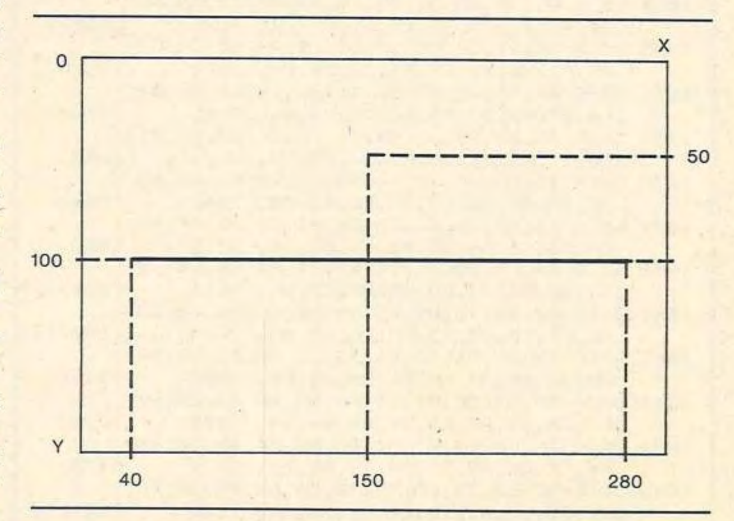


Bild 2. Am Beispiel dieses Bildes wird die Fill-Routine erklärt. Versuchen Sie deshalb, jeden im Text erwähnten Programmschritt an diesem Bild nachzuvollziehen.

Bild 1. Flußdiagramme zur »Fill-Routine«

Das Programm setzt das Füllen der Y-Achse bei X=281 fort, bis der untere Bildschirmrand erreicht ist.

Nun werden die Plotkoordinaten wieder neu gesetzt X=X(3), Y=Y(3) (280, 101), der Feldzeiger um 1 erniedrigt und die beiden Flags FL und FR auf 0 gesetzt (Merken rechter und linker Punkte ist wieder erlaubt).

Die Punkte 1 bis 6 wiederholen sich solange, bis die gesamte Fläche unter der horizontalen Linie gefüllt ist.

Es werden dann nacheinander folgende Flächen gefüllt:
linke Fläche über der Linie bis Anfangskoordinate X=150
linke Bildschirmfläche
rechte Bildschirmfläche

Der Bildschirm wird gefüllt

Weil sich nach dem Füllen der rechten Bildschimfläche keine Koordinate mehr im Feld befindet, der Feldzeiger P also den Wert 0 enthält, wird das Programm beendet.

Sie werden sich fragen, warum diese Fill-Routine extrem schnell ist. Nun, das Programm merkt sich beim Füllen immer die Punkte, ab denen die nächsten Flächen bearbeitet werden. Dabei wird automatisch festgestellt, ob nun von rechts nach links oder von links nach rechts aufgefüllt werden soll.

Mit dieser Beschreibung dürfte es nicht mehr schwer sein, eine eigene Fill-Routine für unterschiedliche Anwendungen, zum Beispiel für Blockgrafik zu realisieren.

Um die Geschwindigkeit der Fill-Routine zu testen, tippen Sie Listing 3 ab oder laden es von Kassette oder Diskette. Gestartet wird es mit RUN. In den ersten zwei Zeilen werden die Adressen der Grafikzeilenanfänge ermittelt und gespeichert. Die Low-Bytes stehen anschließend ab 32256 und die entsprechenden High-Bytes ab 32512. Die restlichen Basic-Zeilen dürften sich selbst erklären.

(Jesko Schwarzer/ah)

Ergänzen Sie jetzt Ihre 64'er-Sammlung

Schaffen Sie sich ein interessantes Nachschlagewerk und gleichzeitig ein wertvolles Archiv!

Kennen Sie alle Ausgaben von 64'er? Suchen Sie einen ganz bestimmten Testbericht? Oder haben Sie einen Teil eines interessanten Kurses versäumt? Suchen Sie nach einer speziellen Anwendung?

Damit Sie jetzt fehlende Hefte mit »Ihrem« Artikel nachbestellen können, finden Sie auf diesen Seiten eine Zusammenstellung aller wesentlichen Artikel der Ausgaben 01 bis 12/85.

Und so kommen Sie schnell an die noch lieferbaren Ausgaben: Prüfen Sie, welche Ausgabe in Ihrer Sammlung noch fehlt, oder welches Thema Sie interessiert. Tragen Sie die Nummer dieser Ausgabe und das Erscheinungsjahr (z.B. 2/85) auf dem Bestellabschnitt der hier eingehefteten Bestell-Zahlkarte ein. Die ausgefüllte Zahlkarte einfach heraustrennen und Rechnungsbetrag beim nächsten Postamt einzahlen. Ihre Bestellung wird nach Zahlungseingang umgehend zur Auslieferung gebracht.

Aktuell

Stichwort	Titel	Seite	Ausgabe
Allgemeines	Commodore Gestern Heute Morgen	10	01/85
Computer	Amiga — Der neue Supercomputer	8	09/85
Interview	Interview mit David Crane (Game Designer)	146	06/85
Lernen	Schule braucht Computer (VAM-Computer)	9	06/85
Messen	International Chaos Communication Congress	15	03/85
	Heiße Messe in der Wüste: CES	8	03/85
	Hannover-Messe '85	8	06/85
	Hannover-Messe '85	8	07/85
	Chicago im Zeichen der CES	8	08/85
	Aktuelles von der C'85 in Köln	15	08/85
	Btx Total (Internationale Funkausstellung)	8	10/85
	PCW-Computermesse in London	8	11/85
	Neues von der Commodore-Fachausstellung 1985	8	12/85
Recht	Die neue Abmahnmasche — Vorsicht bei Programmangeboten	8	05/85
	Die Ex-Knacker — wo sind sie geblieben?	27	08/85
	Interview mit Raubkopierern (Section 8)	28	08/85
	Schützer kontra Knacki's	23	08/85
	Raub-Talkshow	12	08/85
	Das Urheberrechtsgesetz und Gedanken zu seiner Anwendung	21	08/85
	Änderung des Urheberrechtsgesetzes	162	09/85

Buchbesprechungen

Stichwort	Titel	Seite	Ausgabe
Anfänger	Goldmann Computer Compact	87	03/85
	Basic-Wegweiser für den C 64	86	05/85
	Alles über den C 64, Sachbuchreihe, Band 1	115	06/85
	Lehrspielzeug Computer: C 64/VC 20	112	11/85
	C 64 Computerhandbuch	171	11/85
	Einführungskurs: Commodore 64	144	12/85
Anwendung	Dienstprogramme VC 20, C 64 und SX	86	05/85
	Spaß an Mathe mit dem Commodore 64	88	07/85
	Mathe für die Oberstufe mit dem C 64	88	07/85
	Mathematische Routinen VC 20, Elektrotechnik/Elektronik	112	11/85
	Commodore 64-Listings, Band 2: Dateiverwaltung, Schule, Hobby	112	11/85
	Das Trainingsbuch zum Datamat	144	12/85
C 128	Bücher zum C 128	22	10/85
DFÜ	Das Mailbox-Jahrbuch: Nutz die Netze	112	11/85
Grafik	Grafik auf dem Commodore 64 (+Fehlert. 9/85)	86	05/85
	Einführung in CAD mit dem Commodore 64	128	06/85
	Grafik & Musik auf dem Commodore 64	88	07/85
	Verschiedene Grafikbücher zum C 64	115	08/85
Programmieren	Von Basic zu Assembler: Das Commodore-Buch, Band 4	115	06/85
	64 Intern	115	06/85
	Das Interface Age System-Handbuch zum C 64	115	06/85
	Das C 64 Buch, Band 5: Simons Basic Leitfaden	144	12/85
	Basicode	144	12/85
	Noch mehr Tips und Tricks zum 64er	144	12/85
Speichern	Das Kassettenbuch zum C 64 und VC 20	87	03/85
	Die Floppy 1541 (M&T)	88	07/85
Spiele	Rombachs C 64 Spielführer	87	03/85
	Commodore 64-Listings, Band 1, Spiele	112	11/85
	35 ausgesuchte Spiele für Ihren Commodore 64	171	1/85

64'er Extra

Stichwort	Titel	Seite	Ausgabe
Prozessor	Befehlssatz des 6502/6510-Prozessors	84	09/85
Grafik	Die Videochip-Register des C 64	92	10/85
Sound	Der SID-Chip, seine Register und Programmierung	92	11/85
Speicher	Die Speicherbelegung des C 64	96	12/85

Abenteuerlösungen

Stichwort	Titel	Seite	Ausgabe
Lösungen	Dallas-Quest Lösung	90	01/85
	Guncho Krill-Enchanter ist gelöst	44	03/85
	Infocom-Geheimnisse gelüftet?	49	05/85
	Des Rätsels Lösung: Amazon	145	06/85
	Activision-Adventures entschleiert (Mindshadow, Tracer Sanction)	36	12/85
	Eureka! — ich hab's!	37	12/85
	Lösungen zu Hitchhiker's Guide und Sorcerer	39	12/85

Spiele-Tests

Stichwort	Titel	Seite	Ausgabe
007	James Bond — A View to a Kill	156	09/85
Abenteuer	Abenteuerpaket 1	48	08/85
	Shadowfire	146	09/85
	The Quest — mit C 64 auf Suche nach Drachen	47	01/85
Action	Hexenküche	50	07/85
	Master of the Lamps	48	07/85
	Rescue on Fractalus	158	10/85
	Stellar 7	49	08/85
Construction Set	Mail Order Monsters	49	08/85
	Racing Destruction Set	50	08/85
Geschicklichkeit	Australopedicus Robustus	50	08/85
	Boulder Dash II	159	10/85
	Crystal Castles	50	07/85
	Gribbly's Day out	148	09/85
	Rock'n Bolt	48	08/85
	Thing on a Spring	159	10/85
	Tom + Zaga	48	01/85
Pseudo-Adventures	Roland's Rat Race	49	08/85
	Fourth Protocol und Frankie g.t.H.	162	11/85
Renner	Die Renner 1985: Meistverkaufte Spiele	34	12/85
Schach	Viermal Schachmatt: Verschiedene Schachprogramme	32	12/85
Simulation	Elite	148	09/85
	Jump Jet	148	09/85
	Super Huey Hubschraubersimulator	49	07/85
Sport	Boxspiele: Frank Bruno's B. + Barry McGuigan Champions. B.	49	12/85
	Handkantenschlag per Joystick: Karateka + Exploding Fist	165	11/85
	Nick Faldo Plays the Open (Golf)	159	10/85
	Rallye Speedway	49	07/85
	Slapshot (Eishockey)	50	07/85
	Summer Games II	146	09/85
	World Series Baseball	49	07/85
Diverses	New York City und Air Support	145	06/85

Hardware-Tips und Bauanleitungen

Stichwort	Titel	Seite	Ausgabe
Audio/Video	Mit 5 Mark zu neuen Dimensionen (Stereoanlage am C 64)	34	05/85
	Ein Monitor ist genug (RGB+Composite an C 128)	16	10/85
C 16	Alte Datasette am C 16	31	04/85
	Alter Joystick am C 16	35	05/85
Eingabegeräte	Der Hexer — Zusatztastatur für den MSE	48	10/85
EPROM	EPROMs im Expansion-Port	46	10/85
	EPROM-Trans — Die Super-Erweiterung	42	10/85
	Das 64'er EPROM-Programmiergerät, T[illegible]	44	12/85
Floppy/Datasette	Diskettenlaufwerk 1541 selbst justiert	32	10/85
	Die Datasette streikt nie wieder (Anpassung des Tonkopfs)	34	10/85
IEC-Bus	Auf zu neuen Welten: IEC-Bus im Selbstbau (+Fehlerteufel 10/85)	44	07/85
Joystick	Joystick im Selbstbau	33	03/85
	Dauerfeuer-Adapter	46	08/85
RS232/V.24	Das 30-Mark-Interface (Selbstbau RS232)	29	03/85
	Genau betrachtet: Die RS232/V.24-Schnittstelle	80	05/85
Diverses	Userport-Display	36	05/85
	Reset-Taster für alle Fälle (+Fehlert. 9/85)	130	06/85
	Aus eins mach vier (absturzfreie Betriebssystemumschaltung)	41	07/85

Hardware-Grundlagen

Stichwort	Titel	Seite	Ausgabe
Computer	Was bringt der C 128?	28	11/85
Drucker	Welcher Drucker ist der Richtige? (Grundlagen)	15	05/85
	Hammerwerke — wie funktionieren Typenraddrucker	32	06/85
	Die Alternativen: Thermo-, Tintenstrahldrucker + Plotter	24	07/85
Eingabegeräte	Versteht Sie Ihr Computer? (Wie funktionieren Eingabegeräte)	44	09/85
Floppy	Floppy oder Datasette?	129	06/85
Monitore	Wie funktionieren sie, was ist beim Kauf zu beachten?	16	12/85
	Das Kabel zum Monitor: Welche Normen gibt es?	28	12/85
Peripherie	Grafikeingabegerät: Wie funktionieren sie?	30	08/85

Hardware-Tests

Stichwort	Titel	Seite	Ausgabe
Computer	Generationswechsel: Test C 16	16	01/85
	Erster ausführlicher Test C 128 PC (Teil 1)	16	06/85
	Erster ausführlicher Test C 128, PC (Teil 2)	17	07/85
DFÜ	Marktübersicht Modems & Akustikkoppler	32	07/85
Drucker	Vergleich: Drucker unter 700 Mark (Tests und Marktübersicht)	18	05/85
	Tests und Marktübersicht Typenraddrucker	35	06/85
	Test: Brother EP 44	27	07/85
	Brother TC-600	118	08/85
	Riteman C+	133	09/85
	Panasonic KX-P1091	134	09/85
	Star SG 10C	132	09/85
	Melchers CP-80X — wie hätten Sie's denn gern?	25	10/85
	Geheimtip: Der RFI DP 165	24	10/85
	Epson GX 80 — einer für alle	26	10/85
	MPS 803 — ein Drucker für alle Gelegenheiten?	40	1/85
	Epson JX-80 das vielfarbige Druck-Genie	38	11/85
	Epson FX-85 neue Referenz	42	11/85
	SP 1000 VC — Superstar mit Haken	41	11/85
	Der NEC-P2 — das fernöstliche Wunder	159	12/85
	DMPG9 — eine solide Sache	162	12/85
	Das Doppelleben des Joystick-Ports: 10er-Tastaturen	50	09/85
	Joysticks: Test und Marktübersicht (+Fehlerteufel 12/85)	19	11/85
	Es geht auch anders: Lightpens und Trackballs	22	11/85
EPROMer	Frisch gebrannt ist halb gespeichert (EPROM-Programmiergeräte im Test)	39	07/85
	QuickByte II — das Kraftpaket	14	10/85
Floppy/Datasette	Turbo-Floppies, zweite Generation: Speeddos plus + Prologic DOS	28	10/85
	Das große Rennen: Schnelle Bandlaufwerke	37	10/85
	Professionelle Floppylaufwerke für den C 64 (IEC-Floppies)	30	10/85
	Gut gekauft ist halb gespeichert (Marktübersicht Disketten)	38	10/85
Grafik	Die Videowerkstatt (Digitizer-Test)	32	05/85
	Digitalbilder m.d. C 64: PrintTechnik Digitizer	24	01/85
Interface	Hardware-Interface ganz weich: Test EC 64	23	01/85
	Gute Connections — Übersicht Schnittstellen	21	03/85
	Card/Print +6 — Das Allround-Interface	20	03/85
	Das Wiesemann-Centronics-Interface	18	03/85
	Erst ein IEC-Bus öffnet Tür und Tor (+Fehlert.4/6-85)	24	03/85
Monitore	Marktübersicht: Monochrome Monitore	30	12/85
Musik	Trommelwirbel: Test Digital Drums	45	08/85
	Die Musikhardware zum C 64	17	09/85
Roboter	Roboter selbst gebaut (Fischertechnik)	167	10/85
Scanner	So lernt Ihr Drucker lesen	30	06/85
Speicher	Speichertuning VC 20: Test 64 KByte Karte	26	01/85
Steuern	Flottes Türmchen: MEA-Interface	116	08/85

Kurse

Stichwort	Titel	Seite	Ausgabe
Assembler	Assembler ist keine Alchimie, Teil 5	142	01/85
	Assembler ist keine Alchimie, Teil 7	124	03/85
	Assembler ist keine Alchimie, Teil 9	138	05/85
	Assembler ist keine Alchimie, Teil 10	127	07/85
	Assembler ist keine Alchimie, Teil 11	126	08/85
	Assembler ist keine Alchimie, Teil 12	109	09/85
	Assembler ist keine Alchimie, Teil 13 (Schluß)	143	10/85
C 128	Entdeckungsreise duch den C 128	42	12/85
Effektives Programieren	Müllabfuhr im Computer: Garbage Collection, Teil 1	122	01/85
	Finden mit System, eine neuartige Suchmethode, Teil 3	148	03/85
	Sortieren mit dem Computer, Teil 2	159	05/85
	Sortieren mit dem Computer, Teil 3	124	06/85
	Sortieren mit dem Computer, Teil 4	138	08/85
	Sortieren mit dem Computer, Teil 5	124	09/85
	Sortieren mit dem Computer, Teil 6 (Schluß)	150	12/85
Extern	C 64 extern — Der Weg nach draußen, Teil 1	144	08/85
	C 64 extern — Der Weg nach draußen, Teil 2	122	09/85
	C 64 extern — Der Weg nach draußen, Teil 3 (Schluß)	129	10/85
Floppy	In die Geheimnisse der Floppy eingetaucht, Teil 4	148	01/85
	In die Geheimnise der Floppy eingetaucht, Teil 5	130	03/85
	In die Geheimnise der Floppy eingetaucht, Teil 6	145	05/85
	In die Geheimnisse der Floppy eingetaucht, Teil 7 (Schluß)	116	06/85
	Directory-Manipulationen I	140	06/85
Floppy	Directory-Manipulationen II	163	10/85
Grafik	Hires 3 — 15 neue Basic-Befehle, Teil 2	136	03/85
	Hires 3 — Grafikkurs-Anwendung, Teil 3 (Schluß)	152	08/85
	Sprites ohne Geheimnisse	40	08/85
	Streifzüge durch die Grafikwelt, Teil 1	106	09/85
	Streifzüge durch die Grafikwelt, Teil 2	149	11/85
Logeleien	Logeleien, Teil 1	143	07/85
	Logeleien, Teil 2	136	08/85
	Logeleien, Teil 3 (Schluß)	115	09/85
Musik	Dem Klang auf der Spur, Teil 2	136	01/85
	Dem Klang auf der Spur, Teil 4	131	04/85
	Dem Klang auf der Spur, Teil 5	152	05/85
	Dem Klang auf der Spur, Teil 7	132	07/85
	Dem Klang auf der Spur, Teil 8	133	08/85
	Dem Klang auf der Spur, Teil 9	126	10/85
	Dem Klang auf der Spur, Teil 10 (Schluß)	157	11/85
Speicher	Memory Map mit Wandervorschlägen, Teil 3	126	01/85
	Memory Map mit Wandervorschlägen, Teil 5	144	03/85
	Memory Map mit Wandervorschlägen, Teil 7	120	06/85
	Memory Map mit Wandervorschlägen, Teil 8	140	07/85
	Memory Map mit Wandervorschlägen, Teil 9	129	08/85
	Memory Map mit Wandervorschlägen, Teil 10	112	09/85
	Memory Map mit Wandervorschlägen, Teil 11	133	10/85
	Memory Map mit Wandervorschlägen, Teil 12	145	11/85
	Memory Map mit Wandervorschlägen, Teil 13	146	12/85
Sprachen	Basic ist out — es lebe Forth	43	01/85
VC 20	Der gläserne VC 20, Teil 4	130	01/85
	Der gläserne VC 20, Teil 6 (Schluß)	155	03/85

Software-Tips

Stichwort	Titel	Seite	Ausgabe
C 128	Erste Fragen und Antworten zum C 128	14	09/85
	Fragen und Antworten zum 128er	20	10/85
	Fragen und Antworten zum 128er	40	12/85
Drucker	Der MPS 802 lernt Deutsch	30	05/85
	Centronics-Interface für jeden Bedarf	78	07/85
Textverarbeitung	Software Corner — professionelle Programme richtig eingesetzt (Vizawrite-Tips)	174	12/85
Tips & Tricks	Autoboot beim C 64	86	03/85
	Verbindungsfreundlich (Parallelschnittstelle des VC 20)	91	03/85
	Undefinierte Opcodes des 6502	84	03/85
	Durch POKEs zum Erfolg (Spiele-POKEs)	83	03/85
	Tips und Erweiterungen zu Hi-Eddi und Simons Basic	88	03/85
	Basic-Befehle im Griff	79	05/85
	Durch POKEs zum Erfolg: Spiele-POKEs	78	06/85
	Formatierte Eingabe	148	06/85
	Hi-Text (Text in Hires)	70	08/85
	Verbotene Variablen	66	09/85
	Verschiedene Routinen für Anfänger und Profis (+Fehlerteufel 12/85)	88	11/85
	Der Trick mit dem Joystick (Joystickabfrage)	24	11/85
	Verschiedene Tips für Anfänger und Fortgeschrittene	106	12/85

Software-Grundlagen

Stichwort	Titel	Seite	Ausgabe
Assembler	Assembler? Assembler! (Einführung)	32	01/85
	Assembler-Bedienung leicht gemacht, Teil 1	169	12/85
DFÜ	Der erste Kontakt mit DFÜ	40	06/85
	Die Netze der Post: Btx, Datex-P, Telebox	46	06/85
	DFÜ — Was ist das?	44	06/85
	Mailbox für Anfänger	30	07/85

Listings zum Abtippen

Software-Tests

So machen's andere

Die Ausgaben 2/85 und 4/85 sind bereits vergriffen und nicht mehr lieferbar!

```
8:              .opt p2
9:              *=    $7000
10:   ptr1      =     $f7
11:   ptr2      =     $f9
12:   ptr3      =     $fb
13:   sto0      =     $fd
14:   sto1      =     $fe
15:   sto2      =     $ff
16:   xk        =     $55
17:   yk        =     $02
18:             =     $03
19:   msk       =     $04
20:   chkcom    =     $aefd
21:   getcor    =     $b7eb
22:   ;"******************************
23:   ;"*                            *
24:   ;"* byte-paint entwickelt am   *
25:   ;"*                            *
26:   ;"* 25.5.86 von jesko schwarzer *
27:   ;"*                            *
28:   ;"* fuer hires-master.         *
29:   ;"*                            *
30:   ;"* tel.: 02234/62542          *
31:   ;"*                            *
32:   ;"******************************
1000: paint     jsr  gxyk    ;holt anfangskoordinaten
1010:           ldy  #$00
1020:           ldx  #$40    ;anfangsadresse(hb) des
1025:                        ;zeichenspeichers
1030:           lda  #$80    ;hb der 2. page ($8000)
1040:           sty  ptr1
1050:           sty  ptr2
1060:           stx  ptr1+1
1070:           sta  ptr2+1
1080:           ldx  #$20    ;grafikseite nach $8000 kopieren
1090: pl0       lda  (ptr1),y
1100:           sta  (ptr2),y
1110:           iny
1120:           bne  pl0
1130:           inc  ptr1+1
1140:           inc  ptr2+1
1150:           dex
1160:           bne  pl0
1170: ;
1180:           sty  sto0    ;stapelzeiger fuer gemerkte
1185:                        ;punkte auf 0
1190:           jsr  padr2   ;adresse 1. punkt
1200:           and  (ptr1),y ;"punkt gesetzt ?
1210:           beq  pl1     ;nein, dann weiter
1220:           rts          ;ja, dann ende
1230: ;
1240: pl1       sty  sto1    ;merkflag links und
1250: f              sty  sto2   ;rechts zuruecksetzen
1260: ;
1270: pl2       lda  yk      ;oberer rand erreicht
1280:           beq  pl3     ;ja
1290:           jsr  dyk     ;nein, dann schritt nach oben
1300:           lda  (ptr1),y ;"punkt gesetzt ?
1310:           and  msk
1320:           beq  pl2     ;nein, dann anfang schleife
1330:           jsr  iyk     ;auf position unter ges. punkt
1340: ;
1350:           jsr  padr2   ;adresse des punktes und maske
1355:                        ;berechnen
1360:           lda  (ptr1),y ;"ganzes byte leer ?
1370:           beq  pl4     ;ja,>> pl4
1380:           jmp  pl24    ;zu 'bitweise fuellen' springen
1390: ;
1400: pl4       lda  #$ff    ;ganzes byte(8 punkte)
1410:           sta  (ptr1),y ;in $8000 grafik setzen
1420:           lda  yk      ;position in der muster-maske
1430:           and  #$0f    ;berechnen(16*16 bits)
1440:           sta  ptr2    ;(2 reihen mit je 16 bytes)
1450:           lda  xk
1460:           and  #$08
1470:           asl
1480:           adc  ptr2
1490:           tax
1500:           clc
1510:           lda  ptr1
1520:           sta  ptr2
1530:           lda  ptr1+1
1540: of1       adc  #$c0    ;offset zu $8000 grafik addieren
1550:           sta  ptr2+1  ;hb fuer aktuelle grafik
1560:           lda  fmsk,x  ;maskenbyte laden
1570:           sta  (ptr2),y ;und in aktuelle grafik
1575:                        ;schreiben
1580: ; linke seite bearbeiten ********
1590:           lda  xk+1    ;koordinate in linker
1600:           bne  pl5     ;"achter spalte ?
1610:           lda  xk
1620:           cmp  #$08
1630:           bcc  pl9     ;linker rand erreicht
1640: ;
1650: pl5       sec          ;nein,linkes byte errechnen
1660:           lda  ptr1    ;von adresse der plot-position
1670:           sbc  #$08    ;8 abziehen(gleiche zeile,
1680:           sta  ptr2    ;aber linke spalte) und
1690:           lda  ptr1+1  ;nach ptr2
1700:           sbc  #$00
1710:           sta  ptr2+1
1720:           lda  (ptr2),y ;links testen
1730:           bne  pl7     ;kein leeres byte
1740:           lda  sto1    ;"leeres byte, merken erlaubt ?
1750:           bne  pl9     ;merken verboten
1760:           inc  sto1    ;merken erlaubt,aber zukuenftig
1765:                        ;verbieten
1770: ;
1780:           jmp  pl7a
1790: ;
1800: ;merken 1. bit,linke seite
1810: pl7       lsr
1820:           bcs  pl8     ;punkt gesetzt
1830:           lda  sto1    ;"merken erlaubt ?
1840:           bne  pl9     ;merken verboten
1850:           inc  sto1    ;merken erlaubt,aber zuk.
1855:                        ;verbieten
1860: ;
1870: pl7a      ldx  sto0    ;x=x-(xand7)-1
1880:           lda  xk
1890:           and  #$07
1900:           sta  ptr2
1910:           clc
1920:           lda  xk
1930:           sbc  ptr2
1940:           sta  xxl,x   ;low-, high-byte und y-
1950:           lda  xk+1    ;koordinate auf merkstapel
1960:           sbc  #$00    ;ablegen
1970:           sta  xxh,x
1980:           lda  yk
```

Listing 1. Quellcode zur »Fill-Routine«

```
1990:          sta  yyk,x
2000:          cpx  #$ff     ;"schon 255 punkte gemerkt ?
2010:          beq  pl9      ;ja, dann zeiger auf freien
2015:                        ;platz nicht erhoehen
2020:          inc  sto0     ;zeiger naechste position
2030:   ;
2040:          .byt $2c      ;rechte seite
2050:   ;
2060:   pl8    sty  sto1     ;merken wieder erlauben
2070:   ;
2080:   ;rechte seite bearbeiten ********
2090:   pl9    lda  xk+1     ;"rechter rand erreicht ?
2100:          beq  pl10     ;nein >> pl10
2110:          lda  xk
2120:          cmp  #<312
2130:          bcs  pl14     ;ja,naechste zeile bearbeiten
2140:   ;
2150:   pl10   clc           ;rechtes byte
2160:          lda  ptr1     ;byte rechts neben der
2170:          adc  #$08     ;plot-position errechnen
2180:          sta  ptr2     ;(plot-adresse + 8)
2190:          lda  ptr1+1
2200:          adc  #$00
2210:          sta  ptr2+1
2220:          lda  (ptr2),y ;"byte leer ?
2230:          bne  pl12     ;kein leeres byte,merken wieder
2235:                        ;erlauben
2240:          lda  sto2     ;"leeres byte, merken erlaubt ?
2250:          bne  pl14     ;nein, naechste zeile bearb.
2260:          inc  sto2     ;merken zuk. verbieten
2270:   ;
2280:          jmp  pl12a
2290:   ;
2300:   pl12   asl           ;"byte nicht leer, bit 8
2305:                        ;gesetzt ?
2310:          bcs  pl13     ;ja,merkflag zurueck(merken
2315:                        ;erlauben)
2320:          lda  sto2     ;"merken verboten ?
2330:          bne  pl14     ;ja
2340:          inc  sto2     ;merken zuk. verbieten
2350:   ;
2360:   pl12a  ldx  sto0     ;x=(xor7)+1
2370:          clc           ;auf merkstapel
2380:          lda  xk
2390:          ora  #$07
2400:          adc  #$01
2410:          sta  xxl,x
2420:          lda  xk+1
2430:          adc  #$00
2440:          sta  xxh,x
2450:          lda  yk       ;auch die y-koordinate
2460:          sta  yyk,x
2470:          cpx  #$ff     ;"schon 255 punkte ?
2480:          beq  pl14     ;ja, naechste zeile
2490:          inc  sto0     ;nein, naechster freier platz
2500:          .byt $2c      ;naechste zeile
2510:   ;
2520:   pl13   sty  sto2     ;merken im naechsten durchlauf
2525:                        ;erlaubt
2530:   ;
2540:   pl14   jsr  iyk      ;y=y+1 und adresse nach ptr1
2550:          lda  yk       ;schon unterer
2560:          cmp  #$c8     ;"rand ?
2570:          bcs  pl16     ;ja, unterer rand erreicht
2580:          lda  (ptr1),y ;"nein, byte leer ?
2590:          bne  pl18     ;nein, 0-bits merken
2600:          jmp  pl4      ;zum schleifenbeginn
2605:   ;
2620:   pl16   dec  sto0     ;zeigt jetzt auf letzt gemerkten
2625:                        ;punkt
2630:          ldx  sto0
2640:          cpx  #$ff     ;kein punkt mehr, dann ende
2650:          bne  pl17
2660:          rts
2670:   ;
2680:   pl17   lda  xxl,x    ;koordinaten holen
2690:          sta  xk
2700:          lda  xxh,x
2710:          sta  xk+1
2720:          lda  yyk,x
2730:          sta  yk
2740:          jsr  padr2    ;adresse und maske zu den
2745:                        ;koordinaten
2750:          and  (ptr1),y ;"punkt gesetzt ?
2760:          bne  pl16     ;ja, dann naechsten punkt holen
2770:   ;
2780:          jmp  pl1      ;zum schleifenbeginn
2790:   ;
2800:   ;berechnet aus einem byte,welches
2810:   ;sich im akku befindet, die koordinaten
2820:   ;jedes bits und speichert die koord.
2830:   ;aller 0-bits ab
2800:   pl18   cmp  #$ff     ;ganzes byte voll
2810:          beq  pl16     ;keine 0-bits, ende
2820:   ;
2830:          tax           ;nicht leeres byte bearbeiten
2840:          lda  xk       ;koordinaten auf das 8. bit des
2845:                        ;bytes
2850:          and  #$f8
2860:          sta  xk
2870:          sty  f        ;merkflag = 0(yreg.=0!){merken
2875:                        ;im byte erlauben}
2880:   pl19   txa
2890:          and  msk1,y   ;bit-positionen von links nach
2895:                        ;rechts
2900:          bne  pl23     ;gesetztes bit
2910:          lda  f        ;merken erlaubt print
2920:          bne  pl21     ;nein
2930:          inc  f        ;ja, aber zuk. nicht mehr
2940:          txa           ;byte retten
2950:          pha
2960:          ldx  sto0     ;stapelzeiger
2970:          lda  xk       ;koordinaten ablegen
2980:          sta  xxl,x
2990:          lda  xk+1
3000:          sta  xxh,x
3010:          lda  yk
3020:          sta  yyk,x
3030:          cpx  #$ff     ;"schon 255 punkte ?
3040:          beq  pl20     ;ja, dann uebergehen
3050:          inc  sto0     ;merkzeiger erhoehen
3060:   pl20   pla           ;byte zurueckholen
3070:          tax           ;und ins x-reg.
3080:   pl21   inc  xk       ;x-koordinate erhoehen
3090:          iny           ;position im byte erhoehen
3100:          cpy  #$08     ;"schon am ende ?
3110:          bcc  pl19     ;nein, schleife
3120:   ;
3130:          lda  xk       ;x-koordinate erniedrigen
3140:          bne  pl22     ;(um wieder im richtigen
```

```
3150:           dec  xk+1     ;byte zu sein)
3160: pl22      dec  xk
3170:           ldy  #$00     ;y-reg wieder auf null
3180:           jmp  pl16     ;gemerkte punkte bearbeiten
3190:  ;
3200: pl23      lda  #$00     ;merken der 0-bits wieder erl.
3210:           sta  f
3220:           beq  pl21     ;unbedingt
3230:  ;
3240:  ; bit fuellend *****************
3250:  ;wird angesprungen, wenn es nicht
3260:  ;moeglich ist, 'byteweise' 8 punkte
3270:  ;auf einmal zu setzen
3280:  ;
3250: pl24      sty  sto1     ;merkflags fuer linke
3260:           sty  sto2     ;und rechte seite zurueck
3270:  ;
3280: pl25      jsr  padr2    ;adresse und maske(im akku !)
3290:           ora  (ptr1),y ;in $8000 grafik setzen
3300:           sta  (ptr1),y
3310:           clc           ;in aktueller grafik muster
3320:                         ;plotten
3320:           lda  ptr1     ;position in muster-maske
3330:           sta  ptr2     ;berechnen
3340:           lda  ptr1+1
3350: of2       adc  #$c0     ;offset von $8000 grafik zur
3355:                         ;akt.
3360:           sta  ptr2+1
3370:           lda  yk
3380:           and  #$0f
3390:           sta  ptr3
3400:           lda  xk
3410:           and  #$08
3420:           asl
3430:           ora  ptr3
3440:           tax
3450:           lda  fmsk,x   ;byte aus der muster-maske
3460:           and  msk      ;nicht benoetigte bits
3465:                         ;ausmaskieren
3470:           ora  (ptr2),y ;in zeichengrafik schreiben
3480:           sta  (ptr2),y
3490:  ;
3500:           lda  xk       ;"xk = 0 ?
3510:           ora  xk+1
3520:           beq  pl29     ;ja, dann testen links
3525:                         ;ueberspringen
3530:  ;testen links ----
3540:           lda  msk      ;plot-maske
3550:           asl           ;nach links verschieben
3560:           bcc  pl28     ;noch im byte
3570:  ;
3580:           lda  ptr1     ;ausserhalb
3590:           sbc  #$08     ;byteposition neben dem
3595:                         ;'plotbyte'
3600:           sta  ptr2
3610:           lda  ptr1+1
3620:           sbc  #$00
3630:           sta  ptr2+1
3640:           lda  (ptr2),y ;laden
3650:           lsr           ;"1.bit gesetzt ?
3660:           bcs  pl27     ;ja, dann merken erlauben
3670: pl26      lda  sto1     ;"nein, merken links erl. ?
3680:           bne  pl29     ;nein, ueberspringen
3690:           inc  sto1     ;ja, aber jetzt nicht mehr
3700:  ;
3710:           ldx  sto0     ;stapelzeiger
3720:           sec           ;position links speichern
3730:           lda  xk
3740:           sbc  #$01
3750:           sta  xxl,x
3760:           lda  xk+1
3770:           sbc  #$00
3780:           sta  xxh,x
3790:           lda  yk
3800:           sta  yyk,x
3810:           cpx  #$ff     ;"schon 255 ?
3820:           beq  pl29     ;ja, erhoehen ueberspringen
3830:           inc  sto0     ;erhoehen
3840:           jmp  pl29     ;und zur rechten seite
3850:  ;
3860: pl28      and  (ptr1),y ;"bit gesetzt ?
3870:           beq  pl26     ;nein, dann abfrage merken erl.
3875:                         ;usw
3880:  ;
3890: pl27      sty  sto1     ;merken links erlauben
3900:  ;
3910:  ;rechte seite
3920: pl29      lda  xk+1     ;"x=319 ?
3930:           beq  pl30     ;nein, ok
3940:           lda  xk
3950:           cmp  #<319
3960:           bcs  pl34     ;ja, dann rechte seite bearb.
3965:                         ;ueberspr.
3970:  ;
3980: pl30      lda  msk      ;plot-maske
3990:           lsr           ;pixel daneben
4000:           bcc  pl32     ;noch innerhalb des bytes
4010:  ;
4020:           lda  ptr1     ;ausserhalb, 8 addieren(c=1)
4030:           adc  #$07
4040:           sta  ptr2
4050:           lda  ptr1+1
4060:           adc  #$00
4070:           sta  ptr2+1
4080:           lda  (ptr2),y ;byte laden
4090:           asl           ;"bit 8 (punkt) gesetzt ?
4100:           bcs  pl33     ;ja, merken rechts erlauben
4110: pl31      lda  sto2     ;"nein, leer. merken erlaubt ?
4120:           bne  pl34     ;nein, dann merken ueberspringen
4130:           inc  sto2     ;merken verbieten
4140:  ;
4150:           ldx  sto0     ;stapelzeiger
4160:           clc
4170:           lda  xk       ;position rechts neben
4180:           adc  #$01     ;dem zuletzt geplotteten
4190:           sta  xxl,x    ;punkt laden und auf
4200:           lda  xk+1     ;dem stapel ablegen
4210:           adc  #$00
4220:           sta  xxh,x
4230:           lda  yk
4240:           sta  yyk,x
4250:           cpx  #$ff     ;"255 ?
4260:           beq  pl34     ;ja, nicht erhoehen
4270:           inc  sto0     ;nein, naechste freie position
4280:           jmp  pl34     ;naechste zeile
4290:  ;
4300: pl32      and  (ptr1),y ;bit gesetzt print
4310:           beq  pl31     ;nein, >>pl31
4320:  ;
4330: pl33      sty  sto2     ;ja, dann merken erl.
```

```
4340:  ;
4350:  pl34     jsr  iyk     ;down ****************
4360:           lda  yk
4370:           cmp  #$c8     ;"unterer rand erreicht ?
4380:           bcs  pl36    ;ja,gemerkte punkte bearbeiten
4390:           lda  (ptr1),y ;nein,byte laden
4400:           bne  pl35    ;nicht leer,dann bitweise
4405:                        ;bearbeiten
4410:           jmp  pl1     ;sonst schleifenbeginn
4420:  ;
4430:  pl35     and  msk     ;entspr. bit gesetzt
4440:           bne  pl36    ;ja,gemerkte punkte holen
4450:           jmp  pl25    ;nein, dann weiter bitweise
4455:                        ;fuellen
4460:  ;
4470:  pl36     jmp  pl16    ;gemerkte punkte bearbeiten
4480:  ;
4490:  ;
4495:  ;
5000:  padr2    clc          ;adresse und maske berechnen
5010:           ldx  yk      ;y-koordinate ins x-reg.
5020:           lda  xk      ;xlow in akku
5030:           and  #$f8     ;auf beginn des bytes
5040:           adc  aadl,x  ;zeilen beginn low addieren
5050:           sta  ptr1    ;= adresse low
5060:           lda  xk+1    ;xhigh laden
5070:           adc  aadh,x  ;zeilenadresse high addieren
5080:           sta  ptr1+1  ;=adresse high
5090:           lda  xk      ;maske errechnen
5100:           and  #$07     ;xlow and 7 ergibt
5110:           tax          ;7-potenz zur basis 2
5120:           lda  msk1,x  ;aus tabelle laden
5130:           sta  msk     ;und nach msk
5140:           rts
5150:  ;
5160:  ;
5170:  ;
5200:  dyk      dec  yk
5210:           lda  ptr1
5220:           and  #$07
5230:           beq  dykl1
5240:           dec  ptr1
5250:           rts
5260:  dykl1    sec
5270:           lda  ptr1
5280:           sbc  #<313
5290:           sta  ptr1
5300:           lda  ptr1+1
5310:           sbc  #>313
5320:           sta  ptr1+1
5330:           rts
5340:  ;
5350:  ;
5400:  iyk      inc  yk
5410:           lda  ptr1
5420:           and  #$07
5421:           cmp  #$07
5430:           beq  iykl1
5440:           inc  ptr1
5450:           rts
5460:  iykl1    clc
5470:           lda  ptr1
5480:           adc  #<313
5490:           sta  ptr1
5500:           lda  ptr1+1
5510:           adc  #>313
5520:           sta  ptr1+1
5530:           rts
5540:  ;
5550:  ;
6000:  gxyk     jsr  chkcom
6010:           jsr  getcor
6020:           lda  $14
6030:           ldy  $15
6040:           sta  xk
6050:           sty  xk+1
6060:           stx  yk
6070:           rts
8000:  msk1     .byt $80,$40,$20,$10
8010:           .byt $08,$04,$02,$01
8020:  ;
8030:  fmsk     .byt %11111111 ;linke seite
8040:           .byt %10000000
8050:           .byt %10000000
8060:           .byt %10000000
8070:           .byt %10000000
8080:           .byt %10000000
8090:           .byt %10000000
8100:           .byt %10000000
8110:           .byt %11111111
8120:           .byt %10000000
8130:           .byt %10000000
8140:           .byt %10000000
8150:           .byt %10000000
8160:           .byt %10000000
8170:           .byt %10000000
8180:           .byt %10000000
8190:  ;
8200:           .byt %11111111 ;rechte seite
8210:           .byt %10000000
8220:           .byt %10000000
8230:           .byt %10000000
8240:           .byt %10000000
8250:           .byt %10000000
8260:           .byt %10000000
8270:           .byt %10000000
8280:           .byt %11111111
8290:           .byt %10000000
8300:           .byt %10000000
8310:           .byt %10000000
8320:           .byt %10000000
8330:           .byt %10000000
8340:           .byt %10000000
8350:           .byt %10000000
8360:  ;
8900:  xxl      =    $7b00
8910:  xxh      =    $7c00
8920:  yyk      =    $7d00
8930:  ;
8940:  aadl     =    $7e00
8950:  aadh     =    $7f00
8960:  ;
8970:  ;>>> padr2 <<<
8980:  ;unter diesem label wird die adresse
8990:  ;in der $8000 grafik,
8900:  ;die zu xk/xk+1,yk gehoert,errechnet
8910:  ;und in ptr1/ptr1+1 abgelegt.
8920:  ;die bitmaske fuer den entsprechenden
8930:  ;punkt wird nach msk geschrieben
8940:  ;und befindet sich nach abchluss
```

Listing 1. Quellcode zur »Fill-Routine« (Fortsetzung)

```
8950:   ;im akku. das carry ist zu diesem
8960:   ;zeitpunkt nicht gesetzt.
8970:   ;
8980:   ;>>> dyk <<<
8990:   ;die y-koordinate wird um eins ver-
9000:   ;mindert. die zugehoerige adresse
9010:   ;steht in ptr1
9020:   ;
9030:   ;>>> iyk <<<
9040:   ;wie dyk. jedoch wird die y-koord.
9050:   ;erhoeht
9060:   ;
9070:   ;"belegung der zeropage z.b.:
9080:   ;
9090:   ;ptr1 = $f7
9100:   ;ptr2 = $f9
9110:   ;ptr3 = $fb
9120:   ;
9130:   ;sto0 = $fd
9140:   ;sto1 = $fe
9150:   ;sto2 = $ff
9160:   ;
9170:   ;in badh steht das high-byte des
9180:   ;zeichenspeichers.z.b. $40 fuer
9190:   ;die grafik ab $4000
9200:   ;"achtung: die grafik ab $8000
9210:   ;darf nicht benutzt werden, da
9220:   ;sie vom programm aus benoetigt
9230:   ;wird.
9240:   ;
9250:   ;xxl,xxh und yyk sind die merkstapel
9260:   ;sie haben eine laenge von jeweils
9270:   ;256 bytes (insgesammt 768).
9280:   ;sie durfen im freien ram liegen.
9290:   ;da bei hires-master beim aufuehren
9300:   ;die rams mit sei
11370:          lda  #$30
11370:          sta  $01
11380:  ;eingeschaltet werden, liegen sie
11390:  ;dort unter den cias im bereich ab $d000
11400:  ;
11410:  ;die routine mit dem label 'gxyk'
11420:  ;holt die koordinaten und legt
11430:  ;sie in xk/xk+1,yk ab.
```

Listing 1. Quellcode zur »Fill-Routine« (Schluß)

```
10 IF A=0 THEN A=1:LOAD"FILL-C",8,1          <038>
20 SL=32256:SH=32512:AD=2↑15:D=256:FOR I=0
   TO 199 STEP 8                             <223>
30 FOR J=0 TO 7:B=AD+J:H=INT(B/D):L=B-H*D    <069>
40 POKE SL+I+J,L:POKE SH+I+J,H:NEXT:AD=AD+
   320:NEXT                                  <071>
50 REM OBEN WIRD DIE ANFANGSZEILENTABELLE    <209>
60 REM AUFGESTELLT (AADL/H)                  <154>
70 V=53248:POKE V+24,8*16:POKE V+17,59:POK
   E 56576,2:REM  GRAFIK EIN                 <036>
80 GOSUB 140:SYS 12*4096:REM GRAFIK LOESCH
   EN                                        <229>
90 FOR I=6*4096 TO 6*4096+999:POKE I,16:NE
   XT:REM FARBE                              <156>
100 FOR I=0 TO 100:POKE 2↑14+RND(0)*8000,1
    :NEXT                                    <131>
110 SYS 7*4096,160,100:REM PAINT             <230>
120 POKE 198,0:WAIT 198,1                    <092>
130 POKE V+24,21:POKE V+17,27:POKE 56576,3
    :END                                     <160>
140 FOR X=12*4096 TO 12*4096+28:READ A:POK
    E X,A:NEXT:RETURN                        <154>
150 DATA 160,0,132,251,169,64,133,252,152,
    145,251,230,251,208,2,230,252,56,165     <139>
160 DATA 251,233,1,165,252,233,96,144,236,
    96                                       <220>
```

Listing 3. Demoprogramm zu einer der schnellsten »Fill-Routinen« für den C64

```
Name : fill-c                 7000 72f3
---------------------------------------
7000 : 20 b6 72 a0 00 a2 40 a9   95
7008 : 80 84 f7 84 f9 86 f8 85   1b
7010 : fa a2 20 b1 f7 91 f9 c8   1f
7018 : d0 f9 e6 f8 e6 fa ca d0   d0
7020 : f2 84 fd 20 64 72 31 f7   66
7028 : f0 01 60 84 fe 84 ff a5   a1
7030 : 02 f0 0c 20 82 72 b1 f7   24
7038 : 25 04 f0 f3 20 9b 72 20   03
7040 : 64 72 b1 f7 f0 03 4c 8b   b8
7048 : 71 a9 ff 91 f7 a5 02 29   c7
7050 : 0f 85 f9 a5 55 29 08 0a   28
7058 : 65 f9 aa 18 a5 f7 85 f9   8c
7060 : a5 f8 69 c0 85 fa bd cf   ba
7068 : 72 91 f9 a5 56 d0 06 a5   25
7070 : 55 c9 08 90 48 38 a5 f7   8b
7078 : e9 08 85 f9 a5 f8 e9 00   d0
7080 : 85 fa b1 f9 d0 09 a5 fe   18
7088 : d0 33 e6 fe 4c 98 70 4a   6b
7090 : b0 29 a5 fe d0 27 e6 fe   fe
7098 : a6 fd a5 55 29 07 85 f9   26
70a0 : 18 a5 55 e5 f9 9d 00 7b   20
70a8 : a5 56 e9 00 9d 00 7c a5   0a
70b0 : 02 9d 00 7d e0 ff f0 05   0c
70b8 : e6 fd 2c 84 fe a5 56 f0   91
70c0 : 06 a5 55 c9 38 b0 44 18   72
70c8 : a5 f7 69 08 85 f9 a5 f8   75
70d0 : 69 00 85 fa b1 f9 d0 09   3a
70d8 : a5 ff d0 2f e6 ff 4c ea   0c
70e0 : 70 0a b0 25 a5 ff d0 23   0a
70e8 : e6 ff a6 fd 18 a5 55 09   4d
70f0 : 07 69 01 9d 00 7b a5 56   bf
70f8 : 69 00 9d 00 7c a5 02 9d   01
7100 : 00 7d e0 ff f0 05 e6 fd   c6
7108 : 2c 84 ff 20 9b 72 a5 02   62
7110 : c9 c8 b0 07 b1 f7 d0 25   b3
7118 : 4c 49 70 c6 fd a6 fd e0   cc
7120 : ff d0 01 60 bd 00 7b 85   a9
7128 : 55 bd 00 7c 85 56 bd 00   ed
7130 : 7d 85 02 20 64 72 31 f7   83
7138 : d0 e1 4c 2b 70 c9 ff f0   a9
7140 : da aa a5 55 29 f8 85 55   9e
7148 : 84 03 8a 39 c7 72 d0 35   d5
7150 : a5 03 d0 1d e6 03 8a 48   90
7158 : a6 fd a5 55 9d 00 7b a5   24
7160 : 56 9d 00 7c a5 02 9d 00   f5
7168 : 7d e0 ff f0 02 e6 fd 68   93
7170 : aa e6 55 c8 c0 08 90 d2   30
7178 : a5 55 d0 02 c6 56 c6 55   21
7180 : a0 00 4c 1b 71 a9 00 85   06
7188 : 03 f0 e6 84 fe 84 ff 20   a2
7190 : 64 72 11 f7 91 f7 18 a5   f5
7198 : f7 85 f9 a5 f8 69 c0 85   6e
71a0 : fa a5 02 29 0f 85 fb a5   6b
71a8 : 55 29 08 0a 05 fb aa bd   2c
71b0 : cf 72 25 04 11 f9 91 f9   9d
71b8 : a5 55 05 56 f0 41 a5 04   cc
71c0 : 0a 90 36 a5 f7 e9 08 85   4f
71c8 : f9 a5 f8 e9 00 85 fa b1   8b
71d0 : f9 4a b0 29 a5 fe d0 27   23
71d8 : e6 fe a6 fd 38 a5 55 e9   80
71e0 : 01 9d 00 7b a5 56 e9 00   d4
71e8 : 9d 00 7c a5 02 9d 00 7d   61
71f0 : e0 ff f0 0b e6 fd 4c ff   fd
71f8 : 71 31 f7 f0 d7 84 fe a5   07
7200 : 56 f0 06 a5 55 c9 3f b0   07
7208 : 41 a5 04 4a 90 36 a5 f7   a7
7210 : 69 07 85 f9 a5 f8 69 00   65
7218 : 85 fa b1 f9 0a b0 29 a5   dc
7220 : ff d0 27 e6 ff a6 fd 18   8b
7228 : a5 55 69 01 9d 00 7b a5   05
7230 : 56 69 00 9d 00 7c a5 02   6d
7238 : 9d 00 7d e0 ff f0 0b e6   d2
7240 : fd 4c 4a 72 31 f7 f0 d7   8a
7248 : 84 ff 20 9b 72 a5 02 c9   37
7250 : c8 b0 0e b1 f7 d0 03 4c   d5
7258 : 2b 70 25 04 d0 03 4c 8f   fb
7260 : 71 4c 1b 71 18 a6 02 a5   f6
7268 : 55 29 f8 7d 00 7e 85 f7   3a
7270 : a5 56 7d 00 7f 85 f8 a5   f3
7278 : 55 29 07 aa bd c7 72 85   68
7280 : 04 60 c6 02 a5 f7 29 07   73
7288 : f0 03 c6 f7 60 38 a5 f7   f9
7290 : e9 39 85 f7 a5 f8 e9 01   42
7298 : 85 f8 60 e6 02 a5 f7 29   0e
72a0 : 07 c9 07 f0 03 e6 f7 60   74
72a8 : 18 a5 f7 69 39 85 f7 a5   a9
72b0 : f8 69 01 85 f8 60 20 fd   5d
72b8 : ae 20 eb b7 a5 14 a4 15   20
72c0 : 85 55 84 56 86 02 60 80   d7
72c8 : 40 20 10 08 04 02 01 ff   72
72d0 : 80 80 80 80 80 80 80 ff   ce
72d8 : 80 80 80 80 80 80 80 ff   d6
72e0 : 80 80 80 80 80 80 80 ff   de
72e8 : 80 80 80 80 80 80 80 a9   3a
72f0 : 30 85 01 ff 20 36 30 20   d8
```

Listing 2. Übersetzter Maschinencode zur »Fill-Routine«. Bitte mit dem MSE eingeben und speichern.

Kleiner Aufwand, große Wirkung

Wer sich mit Turbo Pascal auskennt, wird vielleicht auch den MOVE-Befehl schätzen und lieben gelernt haben. Dieses Kommando erweitert auch das Basic Ihres C 64 um ein wichtiges und nützliches Hilfsmittel zur Variablenbehandlung und Window-Programmierung.

Wenn Sie diesen Artikel durchgelesen haben, werden Sie sich fragen, warum ein so leistungsfähiger Befehl erst jetzt für das Basic des C 64 entwickelt wurde. Die Routine (MSE-Listing 1 und Assembler-Listing 2) ist zudem so kurz, daß man sie leicht in eigene Programme integrieren kann, was die Möglichkeiten des Programmierers erheblich vergrößert. Die Handhabung des Befehls ist jedoch nicht ganz einfach.

MOVE (VAR1, VAR2, Anzahl) verschiebt ab der Variablen VAR1 »Anzahl« Bytes zur Variablen VAR2. Der MOVE-Befehl wird beim C 64 durch Aufruf mit SYS und nachfolgenden Parametern eingeleitet. VAR1 und VAR2 können hierbei sowohl Variable als auch Adressen von Speicherstellen sein. Die Syntax lautet:

SYS 49152, Adresse1, Adresse2, Anzahl

Der Befehl SYS 49152,1024,50000,1000 verschiebt zum Beispiel den Speicherbereich 1024 bis 1024+1000, also den Bildschirminhalt, nach 50000. Mit SYS 49152,50000,1024,1000 wird der Inhalt des Speicherbereiches 50000 bis 51000 in den Bildschirmspeicher (zurück)geschrieben.

Der erste Befehl kann zum Beispiel in eigenen Programmen eingesetzt werden, um auf Tastendruck den aktuellen Bildschirminhalt zu retten und daraufhin einen Hilfsbildschirm einzublenden. Der zweite SYS-Befehl dient dann dazu, den geretteten Bildschirm zurückzuholen. Das Demoprogramm (Listing 3) zeigt, wie Sie den neuen MOVE-Befehl zur Window-Programmierung einsetzen können. Es wäre genausogut möglich, nur eine einzige Bildschirmzeile, zum Beispiel die oberste, in einen ungenutzten Speicherbereich zu retten, die zum Beispiel eine Menüleiste enthält. Springt man in ein Untermenü, wird dann eine andere Menüleiste mit dem MOVE-Befehl einkopiert. Sie brauchen in diesem Fall den oben beschriebenen SYS-Befehl nur bei »Anzahl« zu ändern (40 Byte (= 1 Zeile) anstelle von 1000 Byte für den gesamten Bildschirmspeicher).

Werden als Adressen Variablennamen angegeben, verwendet die MOVE-Routine die Adressen der zugehörigen Stringdeskriptoren als Quell- und Zieladresse und nicht den Inhalt der Variablen. Sollen also tatsächlich Speicherbereiche verschoben werden, dürfen diese beim SYS-Aufruf nicht als Variablen angegeben werden, sondern unbedingt als numerische, ganzzahlige Werte. Die nachfolgende Befehlsfolge bewirkt somit nicht das gleiche wie der oben beschriebene Befehl zur Verschiebung des Bildschirmspeichers:

```
A=1024:B=50000:C=1000 : SYS 49152,A,B,C
```

In diesem Beispiel werden 1000 Byte ab der Speicherstelle, in der die Variable A gespeichert ist, zur Adresse, ab der B gespeichert war, verschoben. Bei der Angabe von Variablennamen und geeigneter Parameter werden die Inhalte der Variablen selbst verschoben. Wenn in unserem Beispiel die Variable A den Wert 1024 enthält, B den Wert 50000 und C den Wert 1000, dann sucht das Programm zunächst die Variable A, also die Speicherstelle im Basic-RAM, an der »1024« in Fließkommaform abgelegt ist. Ab dieser Speicherstelle (zum Beispiel $0900) werden 1000 Byte zu der Speicherstelle verschoben, ab der der Wert 50000 der Variablen B im Speicher liegt. Die Variable B und der nachfolgende Speicherbereich werden also überschrieben. Richtig eingesetzt, kann der MOVE-Befehl beim Sortieren ausgesprochen nützlich und zeitsparend sein. Angenommen, Sie erstellen eine Dateiverwaltung, die Daten (Datensatzdatei oder Indexdatei) in einem Array sortiert im Speicher hält (zum Beispiel A$(1)="ANDI", A$(2)="BERND" ... A$(20)="WILLI"). Um ein neues Element zum Beispiel mit dem Index zwei einzutragen beziehungsweise einzufügen, müssen die Elemente A$(2) bis A$(20) im Basic 2.0 mit einer Schleife um jeweils ein Element nach unten verschoben werden, um Platz für das neue Element zu schaffen. Um Element drei zu löschen, ist ebenfalls eine Schleife nötig, die die Elemente A$(4) bis A$(20) um je ein Element nach oben verschiebt. MOVE ersetzt beide Schleifen und erledigt das Verschieben beinahe in Nullzeit.

Die Angabe von Variablennamen innerhalb des MOVE-Befehls dürfte nur für die Anwendung auf Arrays interessant sein. Hierbei muß man String-, Integer- und Fließkommavariablen unterscheiden. Zum Verschieben dieser Variablentypen sind unterschiedlich viele Bytes notwendig (Strings: 3 Byte, Integervariablen: 2 Byte und Fließkommavariablen: 5 Byte).

Um in den Arrays A$(1) bis A$(20) (String), A%(1) bis A%(20) (Integer) und A(1) bis A(20) (Fließkomma) jeweils ein neues Element »zwei« einzufügen, sind demnach folgende Befehle notwendig:

```
SYS 49152,A$(2),A$(3),18*3: REM 18 Stringdeskripto-
ren á 3 Byte
SYS 49152,A%(2),A$(3),18*2: REM 18 Integer-
variablen á 2 Byte
SYS 49152,A (2),A (3),18*5: REM 18 Fließkomma-
variablen á 5 Byte
```

Diese Befehle verschieben den Inhalt von jeweils 18 Variablen (beziehungsweise Stringdeskriptoren) eines Arrays (zum Beispiel A$(2) bis A$(19)) um ein Element nach unten (also nach A$(3) bis A$(20)).

Das zweite Demoprogramm (Listing 4) nimmt beliebige Eingaben von der Tastatur entgegen und sortiert diese in alphabetischer Reihenfolge in das Array A$(..) ein. Wenn Sie sich erst in die scheinbar komplizierte Syntax für die Variablenverschiebung eingearbeitet haben, werden auch Sie den MOVE-Befehl nicht mehr missen wollen. Wer den Befehl selbst erweitern oder ändern will, kann sich das kommentierte Assembler-Listing zur Hilfe nehmen. So ist es beispielsweise denkbar, daß die Routine durch geringfügige Änderungen den Variablentyp selbständig erkennt und die Anzahl der notwendigen Bytes für die Verschiebung von Variablen automatisch berechnet. Es wäre auch möglich, die Anfangs- und die Zieladresse beim Verschieben von Speicherbereichen zusätzlich indirekt, durch Variablen, bestimmen zu können. Wir würden uns freuen, wenn Sie uns Ihre Erfahrungen oder selbstentwickelten Änderungen und Erweiterungen zukommen ließen.

(Said Baloui/nj)

```
Name : move.obj                    c000 c0ae
--------------------------------------------
c000 : 20 33 c0 84 fb 85 fc 20  9b
c008 : 33 c0 84 fd 85 fe 20 fd  49
c010 : ae 20 8a ad 20 f7 b7 84  d0
c018 : a7 85 a8 aa a9 00 85 ab  09
c020 : 38 a5 fd e5 fb a5 fe e5  1b
c028 : fc 90 05 e6 ab 20 58 c0  29
c030 : 4c 75 c0 20 fd ae 20 73  28
c038 : 00 a6 7a d0 02 c6 7b c6  16
c040 : 7a c9 30 90 0b c9 3a 10  c5
c048 : 07 20 8a ad 20 f7 b7 60  19
c050 : 20 8b b0 a4 47 a5 48 60  7a
c058 : c0 00 d0 01 ca 88 18 98  ef
c060 : 65 fb 85 fb 8a 65 fc 85  76
c068 : fc 18 98 65 fd 85 fd 8a  5c
c070 : 65 fe 85 fe 60 a0 00 b1  04
c078 : fb 91 fd a5 ab f0 13 a5  4a
c080 : fb d0 02 c6 fc c6 fb a5  7e
c088 : fd d0 02 c6 fe c6 fd 4c  fd
c090 : 9e c0 e6 fb d0 02 e6 fc  7a
c098 : e6 fd d0 02 e6 fe a6 a7  41
c0a0 : d0 02 c6 a8 ca 86 a7 d0  59
c0a8 : ce a5 a8 d0 ca 60 00 ff  3d
```

Listing 1. MOVE-Befehl für den C64. Bitte verwenden Sie zur Eingabe den MSE auf Seite 91.

```
0070;**********************
0080;*   MOVE-ROUTINE     *
0100;* (C) S.BALOUI, 1986 *
0110;**********************
0120;
0130;
0140;*** FUNKTION ***
0150;'MOVE'VERSCHIEBT EINEN BELIEBIGEN
0160;SPEICHERBEREICH, WOBEI DER FALL,
0170;DASS SICH DIE ZIELADRESSE MIT DEM
0180;ZU VERSCHIEBENDEN BLOCK 'UEBER-
0190;LAPPT', BERUECKSICHTIGT WIRD.
0200;
0210;
0220;*** AUFRUF ***
0230;SYS 49152,START,ZIEL,ANZAHL
0240;
0250; - START : ADRESSE ODER VARIABLE
0260; - ZIEL  : ADRESSE ODER VARIABLE
0270; - ANZAHL: INTEGERWERT
0280;
0290;AB 'START' WERDEN 'ANZAHL' BYTES
0300;NACH 'ZIEL' VERSCHOBEN.
0310;BEI ANGABE VON VARIABLEN WIRD
0320;DIE ADRESSE DER ZUGEHOERIGEN
0330;DESCRIPTOREN VERWENDET.
0340;
0350;
0360;*** BEISPIELE ***
0370;1. SYS 49152,1024,50000,1000
0380;   VERSCHIEBT DEN KOMPLETTEN
0390;   BILDSCHIRMSPEICHER NACH
0400;   50000
0410;
0420;2. SYS 49152,A$(10),A$(11),5*3
0430;   START: ADRESSE DES DESCRIPTORS
0440;          AUF 'A$(10)'
0450;   ZIEL:  ADRESSE DES DESCRIPTORS
0460;          AUF 'A$(11)'
0470;   ANZAHL:15 BYTES
0480;
0490;   PRAKTISCHE AUSWIRKUNG:
0500;   IM ARRAY 'A$(1)'..'A$(14)'
0510;   WIRD AN DER POSITION 'A$(10)'
0520;   PLATZ FUER EINEN NEU EINZU-
0530;   TRAGENDEN STRING GESCHAFFEN
0540;   ('A$(10)'..'A$(14)' WERDEN UM
0550;   JE DREI BYTE (EINE DESCRIPTOR-
0560;   LAENGE) VERSCHOBEN.
0570;
0580;
0590CHRGET .DE $73
0600CHRGOT .DE $79
0610TXTPTR .DE $7A
0620VARPOS .DE $B08B
0630CHKKOM .DE $AEFD
0640FRMNUM .DE $AD8A
0650GETADR .DE $B7F7
0660;
0670VON    .DE $FB
0680NACH   .DE $FD
0690ANZAHL .DE $A7
0700KOPIE  .DE $A9
0710FLAG   .DE $AB
0720;
0730       .BA $C000
0740       .OS
0750;
0760;
0770;*** HAUPTPROGRAMM ***
0780       JSR LIES          ;QUELLADRESSE
0790       STY VON           ;LESEN ('VON(+1)')
0800       STA VON+1
0810       JSR LIES          ;ZIELADRESSE
0820       STY NACH          ;LESEN ('NACH(+1)')
0830       STA NACH+1
0840;
0850       JSR CHKKOM        ;ANZAHL LESEN
0860       JSR FRMNUM        ;('ANZAHL(+1)')
0870       JSR GETADR
0880       STY ANZAHL
0890       STA ANZAHL+1
0900;
0910       TAX               ;ANZAHL-HIGH RETTEN
0920;
0930       LDA #0            ;DEFAULT-WERT: 0
0940       STA FLAG          ;('NACH'<'VON')
0950       SEC
0960       LDA NACH          ;'VON(+1)' MIT
0970       SBC VON           ;'NACH(+1)' VER-
0980       LDA NACH+1        ;GLEICHEN
0990       SBC VON+1
1000       BCC P1            ;'NACH'>'VON' =>
1010       INC FLAG          ;FLAG=1
1020       JSR ADRKOR        ;U.ADRESSEN KORRIG.
1030P1     JMP SCHIEB
1040;
1050;
1060;*** PARAMETER LESEN ***
1070LIES   JSR CHKKOM
1080       JSR CHRGET        ;ZEICHEN NACH DEM
1090       LDX TXTPTR        ;KOMMA LESEN
1100       BNE L1            ;TEXTPOINTER WIEDER
1110       DEC TXTPTR+1      ;AUF DAS KOMMA SET-
1120L1     DEC TXTPTR        ;ZEN (DEKREMENT.)
1130;
1140       CMP #'0'          ;VARIABLE, WENN DAS
1150       BCC L2            ;ZEICHEN KEINE ZAHL
1160       CMP #':'          ;IST
1170       BPL L2
1180;
1190       JSR FRMNUM        ;INTEGERWERT LESEN
1200       JSR GETADR
1210       RTS
1220;
1230L2     JSR VARPOS        ;VARIABLENADRESSE
1240       LDY $47           ;HOLEN
1250       LDA $48
1260       RTS
1270;
1280;
1290;*** ADRESS-KORREKTUR ***
1300ADRKOR CPY #0            ;ANZAHL (X/Y)
1310       BNE A1            ;DEKREMENTIEREN =>
1320       DEX               ;X/Y=LOW/HIGH VON
1330A1     DEY               ;ANZAHL-1
1340;
1350       CLC               ;ADRESSEN KORRIG.
1360       TYA               ;'VON(+1)' UND
1370       ADC VON           ;'NACH(+1)' WERDEN
1380       STA VON           ;AUF DAS JEWEILIG.
1390       TXA               ;BLOCKENDE GESETZT,
1400       ADC VON+1         ;INDEM ANZAHL-1
1410       STA VON+1         ;ADDIERT WIRD
1420;                  ;(Y=LOW:ANZAHL-1/
1430       CLC               ; X=HIGH:ANZAHL-1)
1440       TYA
1450       ADC NACH
1460       STA NACH
1470       TXA
1480       ADC NACH+1
1490       STA NACH+1
1500       RTS
1510;
1520;
1530;*** VERSCHIEBE-ROUTINE ***
1540SCHIEB LDY #0            ;Y IMMER NULL
1550S1     LDA (VON),Y       ;EIN BYTE
1560       STA (NACH),Y      ;VERSCHIEBEN
1570;
1580       LDA FLAG          ;'VON'<'NACH' ?
1590       BEQ S4            ;NEIN =>
1600;
1610       LDA VON           ;'VON'<'NACH' :
1620       BNE S2            ;VERSCHIEBEN AB
1630       DEC VON+1         ;BLOCKENDE ABWAERTS
1640S2     DEC VON
1650       LDA NACH          ;BEIDE POINTER
1660       BNE S3            ;DEKREMENTIEREN
1670       DEC NACH+1
1680S3     DEC NACH
1690       JMP S6
1700;
```

```
1710S4   INC VON          ;'VON'>'NACH' :
1720     BNE S5           ;VERSCHIEBEN AB
1730     INC VON+1        ;BLOCKANFANG AUF-
1740S5   INC NACH         ;WAERTS
1750     BNE S6           ;BEIDE POINTER
1760     INC NACH+1       ;INKREMENTIEREN
1770;
1780S6   LDX ANZAHL       ;'ANZAHL' DEKREM-
1790     BNE S7           ;TIEREN UND MIT
1800     DEC ANZAHL+1     ;NULL VERGLEICHEN
1810S7   DEX
1820     STX ANZAHL       ;WENN ANZAHL=NULL :
1830     BNE S1           ;BLOCKVERSCHIEBUNG
1840     LDA ANZAHL+1     ;BEENDET !
1850     BNE S1
1860     RTS
1870     .EN
```

Listing 2. Kommentiertes Assembler-Listing der MOVE-Routine für den C 64

```
100 REM *** MOVE-DEMO ***                          <237>
110 REM *** WINDOWING ***                          <235>
120 :                                              <096>
130 PRINT CHR$(147)                                <159>
140 FOR I=1 TO 20                                  <159>
150 : PRINT "DIES IST EIN TEST"                    <067>
160 NEXT                                           <170>
170 :                                              <146>
180 SYS 49152,1024,50000,1000                      <232>
190 :                                              <166>
200 PRINT CHR$(19)                                 <246>
210 PRINT:PRINT:PRINT                              <079>
220 PRINT TAB(10)"↑******************5"            <203>
230 PRINT TAB(10)"B DIESES WINDOW{3SPACE}B
    "                                              <151>
240 PRINT TAB(10)"B VERSCHWINDET,{3SPACE}B
    "                                              <001>
250 PRINT TAB(10)"B WENN SIE EINE{3SPACE}B
    "                                              <212>
260 PRINT TAB(10)"B TASTE DRUECKEN{2SPACE}
    B"                                             <081>
270 PRINT TAB(10)"B UND WIRD DURCH{2SPACE}
    B"                                             <215>
280 PRINT TAB(10)"B DEN ALTEN BILD- B"             <057>
290 PRINT TAB(10)"B SCHIRMINHALT{4SPACE}B"         <017>
300 PRINT TAB(10)"B ERSETZT{9SPACE}B"              <248>
310 PRINT TAB(10)"Z******************X"            <096>
320 :                                              <042>
330 GET A$:IF A$="" THEN 330                       <205>
340 SYS 49152,50000,1024,1000                      <021>
350 GET A$:IF A$="" THEN 350                       <098>
```

Listing 3. Demoprogramm zur Window-Programmierung mit dem MOVE-Befehl. Beachten Sie die Eingabehinweise auf Seite 91.

```
100 REM *****************                          <237>
110 REM *** MOVE-DEMO ***                          <247>
120 REM *SORTIERTE LISTE*                          <255>
130 REM *****************                          <011>
140 :                                              <116>
150 AM=100                                         <122>
155 DIM A$(AM)                                     <037>
160 PRINT CHR$(147)                                <189>
167 AD=AD+1                                        <184>
170 INPUT"DATENSATZ";DA$                           <241>
180 S=0:E=AD:GOSUB 510:REM BI-SUCHE                <002>
190 SYS 49152,A$(E),A$(E+1),(AM-E)*3               <118>
200 A$(E)=DA$                                      <017>
215 PRINT:PRINT                                    <167>
217 FOR I=1 TO AD:PRINT A$(I):NEXT                 <215>
220 GOTO 167                                       <212>
230 :                                              <206>
240 :                                              <216>
500 REM *** BINAERE SUCHE ***                      <056>
510 M=INT((S+E)/2)                                 <106>
520 IF DA$>A$(M) THEN S=M:GOTO 540                 <237>
530 E=M                                            <148>
540 IF E-S>1 THEN GOSUB 510                        <194>
550 RETURN                                         <100>
```

Listing 4. Demoprogramm zum Verschieben von Variablen innerhalb eines Arrays mit dem MOVE-Befehl

SMON »runderneuert«

Für alle Anwender eines C 64 gibt es jetzt den Promon 64. Bei diesem Programm handelt es sich um eine äußerst leistungsfähige Erweiterung des SMON, die dem Programmierer einen gewaltigen Vorrat an neuen Befehlen zur Verfügung stellt.

Wem der Befehlsvorrat des SMON nicht ausreicht, der braucht »Promon 64«. Hierbei handelt es sich um einen komplett umgeschriebenen und erweiterten SMON, wobei auch einige Verbesserungen am SMON selbst vorgenommen wurden.

Durch den Einbau sämtlicher Zusätze ist der Promon 64 zwar genau doppelt so groß geworden (er belegt 8 KByte im Speicher des Computers), diesen Nachteil macht er aber durch die Leistungsfähigkeit seiner Befehle wieder wett. Wir wollen uns deshalb im folgenden sofort mit dem Befehlssatz des Promon 64 beschäftigen, wobei vorausgesetzt wird, daß Sie den SMON bereits kennen. Bei der Besprechung der Befehle gehen wir in alphabetischer Reihenfolge voran, wobei die Angaben hinter den Befehlsnamen die Parameter bestimmen, die angegeben werden müssen. Selbstverständlich sind beim Eintippen eines Byte oder einer Adresse die Klammern wegzulassen.

A (Start)

Mit diesem Befehl starten Sie den Assembler des Promon 64. Er arbeitet analog zum Assembler des SMON, wobei auch illegale Opcodes eingegeben werden können. Bei einer Fehleingabe springt der Cursor direkt hinter die ausgegebene Adresse und erlaubt es Ihnen damit, den fehlerhaften Text zu überschreiben. Einmal gesetzte Label können bei Änderung ebenfalls wieder neu gesetzt werden.

B (Start) (Ende)

Dieser Befehl erzeugt, wie schon beim SMON, Basic-DATA-Zeilen aus den Speicherinhalten von (Start) bis (Ende). Im Gegensatz zum SMON werden hier jedoch nicht alle Basic-Zeilen ausgegeben, und es wird ein optimierter Algorithmus verwendet, so daß das Erstellen der Zeilen sehr viel schneller geht. Die ungefähre Zeit, die Promon 64 benötigt, wird nach dem Druck auf die <RETURN>-Taste auf dem Bildschirm angezeigt.

C (Start alt) (Ende alt) (Start neu) (Start) (Ende)

Verschiebt einen Speicherbereich von (Start alt) bis (Ende alt) nach (Start neu). Dabei werden alle Sprungadressen im Bereich von (Start) bis (Ende) an den neuen Speicherbereich angepaßt.

D (Start) (Ende)

Startet das Disassemblieren eines Speicherbereichs ab der Adresse (Start). (Ende) ist optional und muß nicht angegeben werden. Wird (Ende) nicht angegeben, so kann die Anzeige der Speicherinhalte durch das Drücken einer beliebigen Taste angehalten und wieder gestartet werden.



E (Adresse)
Mit diesem Befehl kann der Speicherbereich ab (Adresse) mit Hilfe von Zeichen editiert werden. Diese Zeichen werden nach dem Drücken der <F3>-Taste im entsprechenden Speicherbereich als Bildschirmcodes abgelegt. Durch Druck auf <F7> kommen Sie wieder in den Befehlseingabe-Modus zurück.

F (Start) (Ende) (Bytes...)
Die Funktion F sucht im Speicher des C64 ab Adresse (Start) bis (Ende) nach der Bytefolge (Bytes...). Wird die Bytefolge gefunden, so erscheint die Startadresse auf dem Bildschirm.

G (Start)
Startet ein Maschinenprogramm ab der Adresse (Start). Wird keine Adresse angegeben, so nimmt Promon 64 den aktuellen Programmzähler (kann mit R abgefragt werden) als Startadresse.

H (Start) (Ende)
Zeigt den Speicherbereich des Computers von (Start) bis (Ende) an. Dabei werden immer drei Byte nebeneinander als Bitmuster (Bit gesetzt = »*«, Bit gelöscht = ».«) angezeigt, was das Auffinden und Editieren von Sprites ermöglichen soll. (Ende) ist optional, muß also nicht mit angegeben werden.

I (Gerätenummer)
Setzt die Standard-Gerätenummer für alle Disketten-Operationen.

J
Erlaubt das Suchen von Grafikbildern im Speicher des Computers. Dabei wird mit den Tasten <1> bis <8> der gewünschte Speicherbereich angewählt. Die Tasten <H> und <M> erlauben das Umschalten zwischen HiRes- und Multicolor-Darstellung. Durch die Funktionstasten <F1>, <F3> und <F5> können Sie die Bildschirmfarben ändern. Drücken Sie die Taste <SPACE>, so kommen Sie wieder in den Befehlseingabe-Modus zurück.

K (Start) (Ende)
Gibt den Speicherinhalt von (Start) bis (Ende) in ASCII-Codes aus. Das erlaubt das Suchen von Text, wobei (Ende) wieder optional ist.

L "Name" (Start)
Lädt ein Programm von dem Gerät, das mit I angewählt wurde. Normalerweise steht die Geräteadresse auf 8 für die Floppy-Station. (Start) ist optional und kann dazu dienen, ein Programm in einen anderen Speicherbereich zu laden, als der, in dem es normalerweise untergebracht ist.

M (Start) (Ende)
Erlaubt das Anzeigen und Ändern des Speicherinhalts im Bereich von (Start) bis (Ende). Die Anzeige erfolgt zu jeweils acht Byte in einer Zeile und deren ASCII-Codes im Anschluß. Die Angabe von (Ende) ist optional.

N
Holt ein Basic-Programm, das mit NEW gelöscht wurde zurück. Anschließend wird Promon 64 verlassen und ins Basic gesprungen.

O (Start) (Ende) (Byte)
Füllt den Speicherbereich von (Start) bis (Ende) mit dem Wert (Byte).

P (Gerätenummer)
Setzt das Ausgabegerät auf die Nummer (Gerätenummer). Voreingestellt ist hier die Gerätenummer 4 für einen Drucker.

QE (Start) (Ende) (Byte)
Dieser Befehl führt im Bereich von (Start) bis (Ende) ein EOR mit dem angegebenen Wert (Byte) aus und erlaubt so das Codieren großer Speicherbereiche.

QA (Start) (Ende) (Byte)
Siehe QE. Hier wird jedoch eine Addition und kein Exklusiv-ODER ausgeführt.

R
Zeigt die aktuellen Inhalte der Prozessor-Register an. Diese Inhalte können durch Überschreiben auch geändert werden.

S "Name" (Start) (Ende)
Speichert einen Speicherbereich von (Start) bis (Ende) auf das, durch I eingestellte, Gerät. Der voreingestellte Wert für die Gerätenummer beträgt 8.

T (Start) (Byte)
Startet ein Maschinenprogramm an der Adresse (Start) mit dem Inhalt der Speicherstelle $01 (Byte) im »Trace«-Modus. Hierbei wird ein Opcode angezeigt und erst durch Druck auf die <SPACE>-Taste ausgeführt. Anschließend wird der nächste Befehl angezeigt und so weiter. Trifft der Computer auf ein JSR mit Sprung ins Betriebssystem, so kann dieser Aufruf durch Drücken der Taste <J> schnell ausgeführt werden. Danach wird vor dem nächsten Befehl wieder auf den Druck der <SPACE>-Taste gewartet. Der Trace-Befehl kann durch Drücken auf <RUN/STOP> angehalten werden. Er stoppt automatisch beim Erreichen eines BRK- oder RTS-Befehls, der den Programmzähler wieder auf die Ebene des Promon 64 führt. Nach dem Start werden die Prozessor-Register automatisch mit den Werten gefüllt, die mit R abgerufen werden können.

U (Start) (Ende)
Gibt ab der Adresse (Start) alle Speicherinhalte in Bildschirm-Codes aus. Dabei werden keine Adressen angegeben, um den Bildschirmaufbau nicht zu zerstören. Beim Abbruch der Funktion durch <RUN/STOP> wird die gerade erreichte Adresse ausgegeben.

V (Start alt) (Ende alt) (Start neu) (Start) (Ende)
Rechnet alle Sprungadressen im Bereich (Start) bis (Ende) um, wobei keine Verschiebung eines Programms stattfindet. Als Kriterium für die Umrechnung wird der Bereich (Start alt) bis (Ende alt) bezüglich der neuen Adresse (Start neu) genommen. Bitte betrachten Sie dazu auch die Beschreibung des Befehls C.

Eingebauter Disketten-Monitor

W (Start alt) (Ende alt) (Start neu)
Verschiebt ein Programm aus dem Bereich (Start alt) bis (Ende alt) nach (Start neu). Es wird keine Adreßumrechnung vorgenommen.

X
Rückkehr ins Basic. Die Modulkennung des Promon 64 wird dadurch jedoch nicht zerstört, so daß der Monitor durch Druck auf einen Reset-Taster jederzeit wieder aufgerufen werden kann. Eine andere Möglichkeit ist der erneute Aufruf mit SYS 36864.

Y
Zeigt an, ob der Monitor im RAM oder im ROM arbeitet. Siehe dazu die Befehle YA und YO.

YA
Schaltet den Monitor auf RAM-Betrieb. Finden jetzt Operationen statt, die im Bereich des Basic-ROM, des I/O-Bereichs oder des Betriebssystem-ROM liegen, so wird generell der »darunterliegende« RAM-Bereich angesprochen. Das gilt auch für die Befehle L und S.

YO
Hier wird der gesamte Monitor auf ROM-Betrieb umgeschaltet. Es ist jetzt möglich, auch das Basic-ROM oder den Kernel-Bereich zu betrachten, wobei natürlich keine Änderungen der Speicherinhalte möglich sind. Außerdem wird jetzt von $D000 bis $DFFF der I/O-Bereich eingeblendet.

Z
Gibt die Spur- und Sektornummer des zuletzt angesprochenen Sektors auf einer Diskette aus. Wurde der Diskettenmonitor noch nicht verwendet, so erscheint keine Meldung.

8010 Kopiere RAM-Teil für Laden und Speichern
8030 Laderoutine für Disk
8096 Speicherroutine für Disk
80D0 Print Text
80DE Hexcode-Ausgabe zwei Byte
80E2 Hexcode-Ausgabe ein Byte
80F8 Adressenausgabe
8200 Vergleiche Start und Ende
8209 Bildschirmcode-Ausgabe
821F Directory laden
8270 DOS 5.1
82E1 Status/Stop-Abfrage
82F0 ← -Einsprung
830F Eingabe (ohne Punkt)
8317 Ende der Eingabe erreicht?
831F Print CR
8324 Print ASCII und CR
832E Input (ohne Space)
8336 Input (ohne Komma und Space)
833E Input (Fehlerausgabe am Ende)
8345 Einsprung für Fehler
834A Einsprung Hauptmenü
8430 Setze File für Laden oder Speichern
845E L-Einsprung
847F S-Einsprung
84A4 Setze Start und Ende
84B6 Setze Start
84B8 Setze Pointer
84C5 Input ein Byte in Akku
84F0 I-Einsprung
84FB P-Einsprung
8503 Erhöhe Start
850A Stop/Skip/Scroll am Ende der Zeile
8510 Stop/Skip/Scroll an jeder Stelle
8520 Set s.s.s.-Flag, Return und kopiere RAM-Teil
8531 Return und kopiere RAM-Teil
8534 Kopiere RAM-Teil
8551 U-Einsprung
8574 Kontrolliere Gerät und setze Pointer
85A0 Sperre Druckerausgabe
85A5 Setze Start und Ende
85AB Ausgabe auf Monitor und Drucker
85E3 Drucker ausschalten
85F6 Zeile für die Funktion 'H'drucken
8633 H-Einsprung
864F - -Einsprung
8680 Prüfe ASCII
868F K-Einsprung
86CF '-Einsprung
86ED Setze Programmzähler
8700 G-Einsprung
871F Einsprung BRK/RTS
8755 Monitor-Start
8760 R-Einsprung
8793 Leerzeichen und binären Wert drucken
87A6 ;-Einsprung
87C9 Hex-Umrechnung
87ED drucke Hex, Bin, Dec
881B Bin-Umrechnung
883A $-Einsprung
8840 %-Einsprung
8865 Addieren
8873 Subtrahieren
8881 Ende Input?
8897 ?-Einsprung
88F6 Dec-Umrechnung
8947 #-Einsprung
894D X-Einsprung
8956 Zeile für »M« drucken
898D M-Einsprung
89AE ↑-Einsprung
89D0 O-Einsprung
89F1 RAM-Teil löschen und Drucker sperren
8A05 Ausgabe ins RAM
8A17 B-Einsprung
8ADB W-Einsprung
8B62 ↑-Einsprung
8B81 Kopiere RAM-Teil und sperre Drucker
8B87 Lese Tabelle 4 und 5
8BB5 Print CR
8BBA Print CR und ASCII
8BC2 Print Space
8BC7 Fülle Zeile mit Leerzeichen
8BD3 =-Einsprung
8C29 IMP$BDCD
8C30 IMP$A474
8C37 Kopiere RAM-Teil
8C48 Y-Einsprung
8C97 Menü ein (E)
8CC1 Menü aus (E)
8CD9 E-Einsprung
8E73 Drucke und fülle Farb-RAM
8E76 Fülle Farb-RAM
8E93 Start, Ende, Byte Input und sperre Drucker
8EA4 EOR (QE)
8EBD ADD (QA)
8ED6 Q-Einsprung
8EEA N-Einsprung
8F13 Print 4 Leerzeichen
8F16 Print 3 Leerzeichen
8F19 Print 2 Leerzeichen
SF1C Print 1 Leerzeichen

[illegible]F Print Hex-Byte von Opcode
8F5D Print Wort und Adresse
9000 Monitor-Einsprung (SYS36864)
9034 Print ASCII und Leerzeichen
903A D-Einsprung
906B Addiere Länge der Instruktion zu Start
9079 Suche Instruktion
90EC Hex-Code?
9400 Zwei folgende Bytes Hex-Codes? (mit Leerzeichen)
9417 Zwei folgende Bytes Hex-Codes? (ohne Leerzeichen)
941D Zweiter Teil der Befehlssuche
954F Input für Assemblieren
95C4 , -Einsprung
9630 A-Einsprung
97F0 Print Text
9886 Setze RAM-Teil
98C4 Stop/Ende
98D4 Reset-Einsprung (Pointer $8000 bis $8001)
98F6 Setze Pointer
9900 Z-Einsprung
9AA6 Setze Pointer für »V« und »C«
9AC3 Hauptroutine »V«
9B19 V-Einsprung
9B1F C-Einsprung
9B5F F-Einsprung
9BA7 J-Einsprung
9C53 Unterroutine (T)
9C65 T-Einsprung
9E7F Einsprung für Turbo-Tape-Lader
9FFE, 9FFF unbenutzt

Tabelle 1. Hier finden Sie alle wichtigen Einsprungadressen des Promon 64 abgedruckt

ZR (Track) (Sektor)
Liest den Sektor mit der Spurnummer (Track) und der Sektornummer (Sektor) von der Diskette in den Speicher des Computers. Werden die Parameter (Track) und (Sektor) weggelassen, so wird der Sektor eingelesen, dessen Nummer mit Z abgefragt werden kann.

ZN
Liest den logisch nächsten Sektor von der Diskette in den

Speicher des Computers. Auf diese Art und Weise können Files auf der Diskette nachverfolgt werden.

ZW (Track) (Sektor)
Schreibt einen Sektor mit den Parametern (Track) und (Sektor) auf die Diskette. Werden (Track) und (Sektor) weggelassen, so wird der Sektor dahin zurückgeschrieben, von wo er gelesen wurde (Anzeige der Parameter mit Z).

Z (Monitorkommando)
Hier werden Kommandos, die der Promon 64 kann, auf den eingelesenen Sektor bezogen. Es können also beispielsweise mit ZK alle Bytes als ASCII-Zeichen angezeigt werden. Z$ zeigt den Beginn des Sektors im Speicher des Computers an.

@ oder ← oder >
Kennzeichen für DOS 5.1. Hier kann das Directory einer Diskette angezeigt, oder es können Befehle an das Diskettenlaufwerk geschickt werden. Die Syntax der einzelnen Befehle entspricht der des DOS 5.1, also zum Beispiel $ für die Anzeige des Directory.

↑
Dieses Kommando zeigt die Kürzel aller Befehle an, die der Promon 64 eingebaut hat.

(Dezimalzahl)
Rechnet einen Wert (Dezimalzahl) in hexadezimal und binär (falls kleiner als 256) um.

$ (Hexadezimalzahl)
Rechnet einen Wert (Hexadezimalzahl) in dezimal und binär (falls kleiner als 256) um.

% (Binärzahl)
Rechnet eine achtstellige Binärzahl in den entsprechenden Hexadzimal- und Dezimalwert um.

?
Erlaubt das Rechnen mit dem Monitor. Hier können Zahlen der drei Zahlensysteme Hexadezimal, Dezimal oder Binär addiert oder subtrahiert werden, zum Beispiel: ?#23+$10+%00000001. Der Computer gibt dann »$001E 00011110 #30« aus.

= (Start 1) (Ende 1) (Start 2)
Vergleicht den Speicherbereich von (Start 1) bis (Ende 1) mit dem Bereich ab (Start 2). Stimmen Bereiche nicht überein, so werden deren Anfangsadressen auf dem Bildschirm ausgegeben.

£ (Start)
Dieser Befehl sucht auf der Datasette nach einem Programm, das mit Turbo-Tape auf Kassette gespeichert wurde. Wird ein solches Programm gefunden, so wird dessen Name angezeigt. Es kann nun durch Drücken der <SPACE>-Taste das Programm geladen oder durch Druck auf <RUN/STOP> der ganze Vorgang abgebrochen werden. Geben Sie eine Adresse (Start) an, so wird das Programm an diese Adresse geladen. Andernfalls wird es im Speicher an der Stelle abgelegt, von der es gespeichert wurde.

Soweit der Befehlssatz von Promon 64. Das Programm finden Sie unter Listing 1 abgedruckt. Interessiert Sie auch der Aufbau des Promon 64, so hilft Ihnen die Tabelle 1 weiter. Hier sind alle wichtigen Systemadressen des Programms abgedruckt. (Ed van Hout/ks)

```
Name : promon 64             8000 a000
--------------------------------------
8000 : d4 98 5e fe c3 c2 cd 38   92
8008 : 30 20 56 32 20 43 38 36   8e
8010 : 20 37 8c 85 51 85 55 85   41
8018 : 59 60 00 00 00 00 00 78   92
8020 : e6 01 b1 c1 c6 01 60 78   12
8028 : e6 01 91 c1 c6 01 60 02   25
8030 : 86 c1 84 c2 20 10 80 a5   e0
8038 : b9 48 a9 00 85 b9 8d 0e   f8
8040 : dc 20 4a f3 a6 b8 20 0e   0a
8048 : f2 20 13 ee a8 20 13 ee   a3
8050 : aa 68 f0 04 84 c1 86 c2   e1
8058 : a5 90 d0 2f a2 00 20 d0   ac
8060 : 80 a4 c1 a5 c2 20 de 80   01
8068 : a9 00 85 c1 20 13 ee 20   41
8070 : 53 00 a5 90 d0 07 c8 d0   49
8078 : f3 e6 c2 d0 ef 84 ae a2   cc
8080 : 10 20 d0 80 a5 c2 85 af   ca
8088 : 20 de 80 20 33 f3 a5 b8   16
8090 : ee 0e dc 4c 91 f2 86 ae   6e
8098 : 84 af 20 10 80 20 4a f3   18
80a0 : a6 b8 20 50 f2 a4 c1 a9   63
80a8 : 00 85 c1 8d 0e dc 98 20   f7
80b0 : dd ed a5 c2 20 dd ed 20   2e
80b8 : 4b 00 20 dd ed a5 90 d0   b7
80c0 : ca c8 d0 02 e6 c2 c4 ae   58
80c8 : a5 c2 e5 af d0 e9 f0 bb   d5
80d0 : bd 00 81 49 5d f0 06 20   cd
80d8 : 16 e7 e8 d0 f3 60 20 e2   be
80e0 : 80 98 48 4a 4a 4a 4a 20   68
80e8 : ed 80 68 29 0f c9 0a 90   dd
80f0 : 02 69 06 69 30 4c ab 85   75
80f8 : a5 fb a8 a5 fc 4c de 80   28
8100 : 50 11 12 1c 19 14 13 1a   94
8108 : 7d 1b 0f 12 10 7d 79 5d   a7
8110 : 7d 09 12 7d 79 5d 50 50   aa
8118 : 0e 09 12 0d 7d 14 13 7d   91
8120 : 79 5d 50 19 18 0b 14 1e   e6
8128 : 18 7d 13 12 09 7d 0d 0f   d5
8130 : 18 0e 18 13 09 5d 50 14   9d
8138 : 11 11 18 1a 1c 11 7d 19   8e
8140 : 18 0b 14 1e 18 7d 13 08   71
8148 : 10 1f 18 0f 5d 50 50 7d   64
8150 : 7d 0d 1e 7d 7d 0e 0f 7d   0b
8158 : 1c 1e 7d 05 0f 7d 04 0f   8e
8160 : 7d 0e 0d 7d 7d 13 0b 70   55
8168 : 1f 19 14 07 1e 50 66 5d   b3
8170 : 50 0d 11 18 1c 0e 18 7d   1c
8178 : 0a 1c 14 09 7d 5d 7d 0e   8b
8180 : 18 1e 12 13 19 0e 5d 50   a6
8188 : 12 08 09 7d 12 1b 7d 10   a0
8190 : 18 10 12 0f 04 73 5d 50   09
8198 : cc 0d 0f 12 10 12 13 7d   ca
81a0 : 1e 12 10 10 1c 13 19 0e   a8
81a8 : 67 50 50 5d 50 18 0c 08   fd
81b0 : 14 0b 1c 11 18 13 09 50   52
81b8 : 5d 50 0a 12 0f 16 14 13   1a
81c0 : 1a 7d 14 13 7d 0f 5d 10   e6
81c8 : 7d 75 1f 1c 13 16 7d 0a   37
81d0 : 14 09 15 7d 04 1c [illegible] 04   51
81d8 : 12 74 50 5d 50 50 [illegible] 77   38
81e0 : 7d 0d 0f 12 10 12 13 7d   c3
81e8 : 6b 69 7d 77 77 50 1f 04   d5
81f0 : 7d 18 19 7d 0b 1c 13 7d   48
81f8 : 15 12 08 09 73 5d 7d 7d   4d
8200 : a5 fb c5 fd a5 fc e5 fe   ac
8208 : 60 85 d7 a2 00 86 d0 ae   4a
8210 : 86 02 20 13 ea 20 b6 e6   5a
8218 : a5 d8 f0 02 46 d4 60 a9   86
8220 : 01 a6 f8 a0 00 20 74 85   a4
8228 : a0 e8 20 bd ff 20 c0 ff   00
8230 : a2 01 20 c6 ff 20 63 82   c7
8238 : a0 1c 20 cf ff 20 d2 ff   35
8240 : 88 d0 f7 a9 0d 20 d2 ff   80
8248 : 20 e1 82 ea f0 08 a9 01   4f
8250 : 20 c3 ff 4c cc ff 20 63   ef
8258 : 82 48 98 aa 68 20 29 8c   bf
8260 : 4c 38 82 20 cf ff 20 cf   8a
8268 : ff 20 cf ff a8 4c cf ff   97
8270 : a0 00 c9 0d f0 09 99 00   e2
8278 : 02 20 cf ff c8 d0 f3 84   6a
8280 : b6 a9 0d 20 16 e7 a9 01   9b
8288 : a6 f8 a0 6f 20 74 85 a9   cf
8290 : 00 20 bd ff 20 c0 ff a5   63
8298 : b6 f0 1c a5 f8 20 b1 ff   d9
82a0 : a9 6f 20 93 ff a0 00 b9   f4
82a8 : 00 02 20 a8 ff c8 c4 b6   8d
82b0 : d0 f5 a5 f8 20 ae ff a9   ce
82b8 : 00 85 90 a5 f8 20 b4 ff   b7
82c0 : a9 6f 20 96 ff 20 a5 ff   93
82c8 : 85 b6 20 d2 ff 20 a5 ff   a2
82d0 : 20 d2 ff 24 90 50 f6 a5   90
82d8 : f8 20 ab ff a9 01 4c c3   26
82e0 : ff a5 90 d0 0a a2 00 a5   f1
82e8 : 91 c9 7f d0 01 e8 8a 60   9a
82f0 : a9 00 85 90 20 cf ff c9   21
82f8 : 24 d0 10 a9 49 20 70 82   1a
8300 : a5 b6 c9 32 b0 08 20 1f   c3
8308 : 82 a9 0d 4c 70 82 60 20   09
8310 : cf ff c9 2e f0 f9 60 20   b8
8318 : cf ff c6 d3 c9 0d 60 a9   ed
8320 : 0d 4c 16 e7 20 ab 85 a9   9f
8328 : 0d 4c ab 85 00 00 20 cf   17
8330 : ff c9 20 f0 f9 60 20 2e   b9
8338 : 83 c9 2c f0 f9 60 20 cf   8c
8340 : ff c9 0d d0 f8 a9 3f 20   9b
8348 : 16 e7 a6 fa 9a a2 00 86   26
8350 : c6 20 1f 83 a1 d1 c9 27   7d
8358 : f0 15 c9 3a f0 11 c9 3b   c2
8360 : f0 0d c9 2c f0 09 c9 2d   a8
8368 : f0 05 a9 2e 20 16 e7 20   9e
8370 : 3e 83 c9 2e f0 f9 48 29   fa
8378 : 80 85 f9 68 29 7f a2 28   b0
8380 : dd a0 83 f0 05 ca 10 f8   85
8388 : 30 bb 8a 0a 20 92 83 4c   b7
8390 : 4a 83 aa bd d1 83 48 bd   d4
8398 : d0 83 48 60 60 60 60 60   93
83a0 : 40 5f 3e 41 42 43 44 45   21
83a8 : 46 47 48 49 4a 4b 4c 4d   98
83b0 : 4e 4f 50 51 52 53 54 55   a0
83b8 : 56 57 58 59 5a 2d 27 3b   1d
83c0 : 24 25 23 3f 3a 5e 3d 2c   0b
83c8 : 5c a9 80 85 38 4c 44 a6   0e
83d0 : ef 82 ef 82 ef 82 2f 96   49
83d8 : 16 8a 1e 9b 39 90 d8 8c   c3
83e0 : 5e 9b ff 86 32 86 ef 84   fd
83e8 : a6 9b 8e 86 5d 84 8c 89   0f
83f0 : e9 8e cf 89 fa 84 d5 8e   8e
83f8 : 5f 87 7e 84 64 9c 50 85   c2
8400 : 18 9b da 8a 4c 89 47 8c   35
8408 : ff 98 4e 86 ce 86 a5 87   7e
8410 : 39 88 3f 88 46 89 96 88   8a
8418 : ad 89 61 8b d2 8b c3 95   17
8420 : 7e 9e a9 10 24 01 d0 02   eb
8428 : 24 01 18 60 00 00 00 00   df
8430 : 20 2e 83 c9 22 d0 0f a0   a7
8438 : 00 20 3e 83 c9 22 f0 09   cc
8440 : 99 00 02 c8 d0 f3 4c 45   db
8448 : 83 c0 00 f0 f9 84 b7 a9   3f
8450 : 00 85 bb a9 02 85 bc 85   81
8458 : b8 a6 f8 4c 49 86 20 30   d5
8460 : 84 a9 01 85 b9 20 17 83   aa
8468 : f0 07 c6 b9 a2 c1 20 b8   ef
8470 : 84 20 34 80 a5 90 c9 40   a8
8478 : f0 3b a9 0d 4c 70 82 20   a4
8480 : 30 84 a9 01 85 b9 20 17   52
8488 : 83 f0 bb a2 c1 20 b8 84   cf
8490 : a2 ae 20 b8 84 e6 ae d0   84
8498 : 02 e6 af 20 9a 80 a5 90   63
84a0 : f0 13 d0 d6 20 b6 84 a9   46
84a8 : ff 85 fd 85 fe 20 17 83   ee
84b0 : f0 03 20 b8 84 60 a2 fb   0f
84b8 : 20 c5 84 95 01 20 d2 84   f4
84c0 : 95 00 e8 e8 60 20 3e 83   b3
84c8 : c9 20 f0 f9 c9 2c f0 f5   ca
84d0 : d0 03 20 3e 83 20 e7 84   d4
84d8 : 0a 0a 0a 0a 85 b4 20 3e   a6
84e0 : 83 20 e7 84 05 b4 60 c9   09
```

Listing 1.
Das Programm »Promon 64« geben Sie bitte mit dem MSE ein.
Beachten Sie dazu unsere Eingabehinweise auf der Seite 92 dieser Ausgabe.

```
84e8 : 3a 90 02 69 08 29 0f 60  df
84f0 : 20 c5 84 f0 03 85 f8 60  33
84f8 : 4c 45 83 20 c5 84 f0 f8  02
8500 : 85 f7 60 e6 fb d0 02 e6  92
8508 : fc 60 a5 d3 c9 27 d0 1c  69
8510 : 20 3e f1 f0 11 c9 20 d0  6b
8518 : 04 a9 01 85 b6 a5 b6 49  e8
8520 : 01 85 b6 4c ed f6 a5 b6  b6
8528 : f0 e6 d0 f7 60 a9 00 85  1d
8530 : b6 20 1f 83 20 86 98 85  d2
8538 : 51 85 56 85 5a ea 60 78  02
8540 : e6 01 b1 fb c6 01 58 60  29
8548 : 78 e6 01 91 fb c6 01 58  50
8550 : 60 20 a5 85 20 2d 85 a0  9d
8558 : 00 20 4b 00 20 09 82 20  d0
8560 : 00 82 b0 08 20 03 85 20  3f
8568 : 0a 85 d0 eb a2 16 20 d0  e3
8570 : 80 4c f8 80 85 b8 84 b9  08
8578 : 48 98 48 8a d0 04 a9 08  54
8580 : 85 f8 85 ba a2 00 86 90  9f
8588 : 20 b1 ff 20 ae ff a5 90  27
8590 : f0 08 a2 22 20 d0 80 4c  94
8598 : 4a 83 a2 08 68 a8 68 60  7c
85a0 : a9 00 85 f9 60 20 a0 85  7e
85a8 : 4c a4 84 48 a5 f9 0a 90  e4
85b0 : 2e 29 02 d0 25 a6 f7 d0  16
85b8 : 04 a2 04 86 f7 e0 08 90  a7
85c0 : 08 a2 36 20 d0 80 4c 4a  82
85c8 : 83 a9 04 a0 00 20 74 85  13
85d0 : 20 4a f3 a6 b8 20 50 f2  9b
85d8 : e6 f9 68 48 20 dd ed 68  57
85e0 : 4c 16 e7 a5 f9 29 01 f0  b5
85e8 : 08 20 33 f3 a9 04 20 91  aa
85f0 : f2 a9 00 85 f9 60 a9 2d  0b
85f8 : 20 ab 85 20 f8 80 a0 00  69
8600 : a9 20 20 ab 85 20 4b 00  bd
8608 : 85 c1 a2 08 06 c1 90 04  d0
8610 : a9 2a d0 02 a9 2e 20 ab  27
8618 : 85 ca d0 f0 c8 c0 03 d0  95
8620 : e4 a5 fb 18 69 03 85 fb  95
8628 : a5 fc 69 00 85 fc a9 0d  a7
8630 : 4c ab 85 20 a4 84 20 2d  00
8638 : 85 20 f6 85 20 00 82 b0  a9
8640 : 05 20 10 85 d0 f3 4c e3  b0
8648 : 85 20 78 85 4c a0 85 20  cc
8650 : b6 84 20 a0 85 20 34 85  9a
8658 : 20 3e 83 a0 00 a2 08 a9  15
8660 : 00 85 c1 20 3e 83 c9 2a  13
8668 : d0 06 38 26 c1 4c 72 86  64
8670 : 06 c1 ca d0 ee a5 c1 20  87
8678 : 54 00 c8 c0 03 d0 de 60  09
8680 : c9 20 90 08 c9 a0 b0 06  ef
8688 : c9 80 90 02 a9 2e 60 20  c4
8690 : a4 84 20 2d 85 a9 27 20  a7
8698 : ab 85 20 f8 80 a9 20 20  43
86a0 : ab 85 a0 00 20 4b 00 20  d3
86a8 : 80 86 20 ab 85 c8 c0 20  cb
86b0 : d0 f2 a9 27 20 24 83 a5  c5
86b8 : fb 18 69 20 85 fb 90 02  9c
86c0 : e6 fc 20 00 82 b0 05 20  2e
86c8 : 10 85 d0 c9 4c e3 85 20  42
86d0 : b6 84 20 a0 85 20 34 85  1a
86d8 : 20 3e 83 a0 00 20 3e 83  0d
86e0 : c9 2e f0 03 20 54 00 c8  93
86e8 : c0 20 d0 f1 60 20 17 83  95
86f0 : f0 0d 20 b6 84 a5 fb 8d  c6
86f8 : 0c 01 a5 fc 8d 0b 01 60  84
8700 : 20 f1 89 20 ed 86 a6 fa  23
8708 : 9a a9 87 48 a9 1d 48 a2  4c
8710 : fa bd 11 00 48 e8 d0 f9  30
8718 : 68 a8 68 aa 68 40 00 d8  7e
8720 : a9 00 8d 20 d0 8d 21 d0  d0
8728 : a9 03 8d 86 02 a2 05 68  a1
8730 : 9d 0b 01 ca 10 f9 68 c9  f2
8738 : 1d d0 06 68 c9 87 f0 02  ed
8740 : 48 48 ad 0c 01 d0 03 ce  d9
8748 : 0b 01 ce 0c 01 ba 86 fa  ff
8750 : a9 52 4c 7e 83 a9 1f 8d  22
8758 : 16 03 a9 87 8d 17 03 00  e9
8760 : a2 4d 20 d0 80 ad 0b 01  6f
8768 : 20 e2 80 ad 0c 01 20 e2  de
8770 : 80 a9 20 20 16 e7 a2 fc  f6
8778 : bd 11 00 20 e2 80 a9 20  db
8780 : 20 16 e7 e8 d0 f2 a5 fa  f3
8788 : 20 e2 80 a9 20 20 16 e7  9a
8790 : ad 0d 01 85 aa a9 20 a0  6f
8798 : 09 20 16 e7 06 aa a9 30  f1
87a0 : 69 00 88 d0 f4 60 20 b6  86
87a8 : 84 8d 0c 01 a5 fc 8d 0b  a5
87b0 : 01 a2 fb 20 3e 83 20 c5  11
87b8 : 84 9d 12 00 e8 d0 f4 85  83
87c0 : fa a9 20 20 16 e7 4c 90  8e
87c8 : 87 20 2e 83 c9 0d d0 03  aa
87d0 : 4c 45 83 c6 d3 a9 00 85  0e
87d8 : fc 20 c5 84 85 fb 20 81  a2
87e0 : 88 f0 09 a5 fb 85 fc 20  f7
87e8 : c5 84 85 fb 60 a9 00 85  2e
87f0 : d3 85 f9 20 6c e5 a9 24  ed
87f8 : 20 16 e7 20 f8 80 a5 fc  45
8800 : d0 05 a5 fb 20 93 87 a9  4c
8808 : 20 20 16 e7 a9 23 20 16  1b
8810 : e7 a6 fb a5 fc 4c 29 8c  ee
8818 : a9 00 85 fb 85 fc a0 08  75
8820 : 20 3e 83 c9 30 d0 05 06  23
8828 : fb 4c 36 88 c9 31 f0 03  d8
8830 : 4c 45 83 38 26 fb 88 d0  0d
8838 : e7 60 20 c9 87 4c ed 87  32
8840 : 20 18 88 4c ed 87 20 3e  30
8848 : 83 c9 20 f0 f9 c9 24 d0  f6
8850 : 03 4c c9 87 c9 25 d0 03  ec
8858 : 4c 18 88 c9 23 d0 03 4c  69
8860 : f6 88 4c 45 83 a5 fd 18  e4
8868 : 65 fb 85 fd a5 fe 65 fc  ce
8870 : 85 fe 60 a5 fd 38 e5 fb  72
8878 : 85 fd a5 fe e5 fc 85 fe  9f
8880 : 60 20 2e 83 c6 d3 c9 2b  75
8888 : f0 0c c9 2d f0 08 c9 0d  27
8890 : d0 04 a9 00 85 c2 60 a2  02
8898 : 00 86 fd 86 fe 86 f9 86  45
88a0 : b6 e8 86 c2 20 46 88 a5  66
88a8 : c2 d0 03 4c 45 83 a5 fb  1c
88b0 : 85 fd a5 fc 85 fe 20 2e  6a
88b8 : 83 c9 2b f0 1a c9 2d f0  8f
88c0 : 16 c9 0d d0 f1 a6 b6 f0  29
88c8 : e2 20 16 e7 a5 fd 85 fb  95
88d0 : a5 fe 85 fc 4c ed 87 85  53
88d8 : c1 20 46 88 e6 b6 a5 c1  8a
88e0 : c9 2b d0 0c 20 65 88 a5  8f
88e8 : c2 d0 cb a9 0d 4c c9 88  a6
88f0 : 20 73 88 4c e7 88 20 3e  35
88f8 : 83 c9 20 f0 f9 c6 d3 a2  f0
8900 : 00 8a 86 fb 85 fc a8 20  89
8908 : cf ff c9 0d d0 09 a9 00  e7
8910 : 85 c2 86 fb 84 fc 60 c9  5d
8918 : 3a b0 04 e9 2f b0 04 c6  ff
8920 : d3 d0 ef 85 ae 06 fb 26  5f
8928 : fc a5 fc 85 af a5 fb 0a  13
8930 : 26 af 0a 26 af 18 65 fb  be
8938 : 08 18 65 ae aa a5 af 65  dd
8940 : fc 28 69 00 4c 02 89 20  e6
8948 : f6 88 4c ed 87 a6 fa 9a  22
8950 : 20 5b ff 4c ce 98 a9 3a  74
8958 : 20 ab 85 20 f8 80 a0 00  c9
8960 : a9 20 20 ab 85 20 4b 00  1d
8968 : 20 e2 80 c8 c0 08 d0 f0  a4
8970 : a0 02 a9 20 20 ab 85 88  06
8978 : d0 fa 20 4b 00 20 80 86  47
8980 : 20 ab 85 c8 c0 08 d0 f2  66
8988 : a9 0d 4c ab 85 [illegible] [illegible] 84  35
8990 : 20 2d 85 20 56 89 a5 fb  ec
8998 : 18 69 08 85 fb 90 02 e6  32
89a0 : fc 20 00 82 b0 05 20 10  d1
89a8 : 85 d0 e8 4c e3 85 20 81  47
89b0 : 8b 20 b6 84 a0 00 20 3e  90
89b8 : 83 20 c5 84 20 54 00 c8  83
89c0 : c0 08 d0 f2 a9 00 85 d3  6f
89c8 : 20 6c e5 c6 d6 4c 56 89  ad
89d0 : 20 b6 84 20 b8 84 20 c5  2c
89d8 : 84 85 b6 20 31 85 a0 00  92
89e0 : a5 b6 20 54 00 20 00 82  79
89e8 : b0 06 20 03 85 4c e0 89  55
89f0 : 60 a2 11 a9 00 95 4a ca  86
89f8 : d0 fb a9 03 85 53 a9 4c  c3
8a00 : 85 54 86 f9 60 84 b6 a0  d6
8a08 : 00 91 2d e6 2d d0 02 e6  28
8a10 : 2e e6 b8 a4 b6 18 60 20  62
8a18 : b6 84 20 b8 84 20 31 85  49
8a20 : 20 73 88 f0 17 85 b8 4a  4f
8a28 : 65 b8 48 a2 70 20 d0 80  9c
8a30 : 68 aa a9 00 20 29 8c a2  1a
8a38 : 7e 20 d0 80 a9 00 85 f9  af
8a40 : 8d 0e dc 85 ae 20 65 88  4f
8a48 : a9 05 8d 26 03 a9 8a 8d  5f
8a50 : 27 03 a5 2d 38 e9 02 85  ee
8a58 : c1 a5 2e e9 00 85 c2 a9  3f
8a60 : 7d 85 af a0 00 84 b8 a5  f2
8a68 : ae 20 05 8a a5 af 20 05  1b
8a70 : 8a e6 ae d0 02 e6 af a9  9c
8a78 : 83 20 05 8a a0 00 20 4b  bf
8a80 : 00 aa 98 20 29 8c 20 00  77
8a88 : 82 b0 3f 20 03 85 a5 b8  9b
8a90 : c9 45 b0 08 a9 2c 20 05  af
8a98 : 8a 4c 7c 8a 20 a8 8a 90  4b
8aa0 : c2 a2 87 20 d0 80 f0 25  b8
8aa8 : a0 00 98 20 05 8a a5 2d  08
8ab0 : 48 91 c1 c8 a5 2e 91 c1  e0
8ab8 : 85 c2 68 85 c1 a9 00 20  13
8ac0 : 05 8a 88 10 fa a5 2e c9  58
8ac8 : 7f 60 20 a8 8a a9 ca 8d  d1
8ad0 : 26 03 a9 f1 8d 27 03 ee  1c
8ad8 : 0e dc 60 20 b6 84 20 b8  f2
8ae0 : 84 a2 c1 20 b8 84 20 34  c2
8ae8 : 85 a9 00 8d 0e dc 85 f9  c5
8af0 : 20 73 88 a5 c1 18 65 fd  0f
8af8 : 85 c3 a5 c2 65 fe 85 c4  0e
8b00 : 20 65 88 a5 c1 c5 fb a5  2f
8b08 : c2 e5 fc 90 38 a5 c1 c5  51
8b10 : fd a5 c2 e5 fe f0 02 b0  2e
8b18 : 2c a9 fd 85 4f a9 c3 85  a5
8b20 : 58 a0 00 20 4b 00 20 54  aa
8b28 : 00 20 00 82 b0 13 a5 fd  bf
8b30 : d0 02 c6 fe c6 fd a5 c3  0d
8b38 : d0 02 c6 c4 c6 c3 4c 23  55
8b40 : 8b ee 0e dc 60 a0 00 a9  c0
8b48 : c1 85 58 20 4b 00 20 54  c4
8b50 : 00 20 00 82 b0 eb 20 03  a2
8b58 : 85 e6 c1 d0 ee e6 c2 4c  a4
8b60 : 4b 8b a2 97 20 d0 80 a0  d8
8b68 : 00 b9 a0 83 f0 0d 20 16  01
8b70 : e7 a2 09 a9 20 20 16 e7  4b
8b78 : ca d0 fa c8 c0 29 d0 e9  ee
8b80 : 60 20 a0 85 4c 34 85 a2  8b
8b88 : 00 86 b7 86 b8 86 ba 86  42
8b90 : bb aa bd 00 92 85 bc d0  fa
8b98 : 01 60 bd 00 91 aa 29 03  52
8ba0 : 85 b7 8a 29 1c 4a 4a 85  11
8ba8 : b9 8a 0a 26 ba 0a 26 b8  f4
8bb0 : 0a 26 bb 60 60 a9 0d 4c  e8
8bb8 : ab 85 48 20 b5 8b 68 4c  2e
8bc0 : ab 85 a9 20 4c ab 85 a4  1e
8bc8 : d3 a9 20 91 d1 c8 c0 28  61
8bd0 : d0 f9 60 20 b6 84 20 b8  3a
8bd8 : 84 a2 c1 20 b8 84 20 1f  90
8be0 : 83 a2 0d bd ed 8b 95 4a  d5
8be8 : ca d0 f8 4c 8b 8e 78 e6  bf
8bf0 : 01 b1 fb d1 c1 08 c6 01  7c
8bf8 : 58 28 60 86 b6 a0 00 20  fe
8c00 : 4b 00 f0 08 20 f8 80 20  95
8c08 : c2 8b 85 b6 20 c4 98 b0  b4
8c10 : 0c 20 03 85 e6 c1 d0 e5  29
8c18 : e6 c2 4c fd 8b a5 b6 d0  94
8c20 : 05 a2 ac 20 d0 80 4c e3  af
8c28 : 85 a0 37 84 01 4c cd bd  81
8c30 : a9 37 85 01 4c 74 a4 a2  37
8c38 : 10 bd 1e 80 95 4a ca d0  37
8c40 : f8 4c 90 98 00 00 00 00  96
8c48 : 20 cf ff c9 0d d0 19 a2  8a
8c50 : b9 20 d0 80 a5 02 c9 36  5b
8c58 : f0 04 a9 41 d0 02 a9 4f  3f
8c60 : 20 16 e7 a2 c7 4c d0 80  fd
8c68 : a2 36 c9 41 f0 07 c9 4f  cd
8c70 : f0 04 4c 45 83 e8 86 02  bc
8c78 : 60 a0 86 b1 a0 85 84 89  5b
8c80 : 94 a0 a0 a0 86 b3 a0 93  50
8c88 : 94 8f 92 85 a0 a0 86 b7  d2
8c90 : a0 91 95 89 94 a0 a0 a2  a5
8c98 : 09 bd 77 05 9d 11 01 bd  e0
8ca0 : 79 8c 9d 77 05 bd 9f 05  7c
8ca8 : 9d 1b 01 bd 83 8c 9d 9f  1d
8cb0 : 05 bd c7 05 9d 25 01 bd  a9
8cb8 : 8d 8c 9d c7 05 ca 10 d9  86
8cc0 : 60 a2 09 bd 11 01 9d 77  ea
8cc8 : 05 bd 1b 01 9d 9f 05 bd  f9
8cd0 : 25 01 9d c7 05 ca 10 eb  95
8cd8 : 60 20 a5 85 a9 93 20 73  01
8ce0 : 8e a5 fb 85 c1 18 69 e7  43
8ce8 : 85 fd a5 fc 85 c2 69 03  8f
8cf0 : 85 fe a5 d1 85 ae a5 d2  a2
8cf8 : 85 af 20 34 85 a0 00 20  81
8d00 : 4b 00 91 d1 20 00 82 b0  57
8d08 : 0c 20 03 85 e6 d1 d0 ef  b6
8d10 : e6 d2 4c ff 8c a9 13 20  15
8d18 : 16 e7 20 97 8c a2 20 a0  bc
8d20 : 00 a5 c5 c9 04 f0 10 c9  39
8d28 : 05 f0 0c c9 03 f0 08 c8  4b
8d30 : d0 ef e8 d0 ea a9 00 85  53
8d38 : b6 20 c1 8c a5 b6 d0 0d  6e
8d40 : a2 00 a0 00 c8 d0 fd e8  e7
8d48 : d0 f8 4c 1a 8d c9 03 d0  c0
8d50 : 05 a9 93 4c 16 e7 c9 05  6a
8d58 : d0 25 a9 13 20 16 e7 a5  25
8d60 : c1 85 fb a5 c2 85 fc a0  25
8d68 : 00 b1 d1 20 54 00 20 00  7f
8d70 : 82 b0 a2 20 03 85 e6 d1  93
8d78 : d0 ef e6 d2 4c 69 8d 4c  33
8d80 : be 8d a0 00 b1 d1 49 80  fd
8d88 : 91 d1 a5 cf 49 01 85 cf  b7
8d90 : a9 11 85 cd 60 a0 00 84  f1
8d98 : cf 84 ce 84 cd 20 42 f1  b8
8da0 : f0 0a 48 a5 cf f0 03 20  2d
8da8 : 82 8d 68 60 e6 ce d0 ed  1b
8db0 : a5 cd c6 cd c9 00 d0 e5  53
8db8 : 20 82 8d 4c 9d 8d 20 95  f8
8dc0 : 8d c9 0d d0 03 4c 15 8d  91
8dc8 : c9 11 d0 15 a5 d6 c9 18  59
8dd0 : f0 ec e6 d6 a5 d1 18 69  e7
8dd8 : 28 85 d1 90 e1 e6 d2 d0  8b
8de0 : dd c9 91 d0 13 a5 d6 f0  bc
8de8 : d5 c6 d6 a5 d1 38 e9 28  61
8df0 : 85 d1 b0 ca c6 d2 d0 c6  b7
8df8 : c9 1d d0 1c a5 d3 c9 27  76
8e00 : d0 0c a5 d6 c9 18 f0 b6  a9
```

Listing 1. »Promon 64« (Fortsetzung)

```
8e08 : e6 d6 a9 ff 85 d3 e6 d3  fe
8e10 : e6 d1 d0 aa e6 d2 d0 a6  fe
8e18 : c9 9d d0 1b a5 d3 d0 0a  98
8e20 : a5 d6 f0 9a c6 d6 a9 28  da
8e28 : 85 d3 c6 d3 a5 d1 d0 02  f3
8e30 : c6 d2 c6 d1 4c be 8d c9  d0
8e38 : 20 b0 03 4c be 8d c9 40  fb
8e40 : 90 16 c9 60 90 0c c9 80  eb
8e48 : 90 0b c9 a0 90 ed c9 c0  05
8e50 : b0 0d 38 e9 20 38 e9 20  7e
8e58 : a0 00 91 d1 4c fc 8d c9  0d
8e60 : e0 b0 06 38 e9 60 4c 37  62
8e68 : 8e c9 ff f0 ce 38 e9 e0  11
8e70 : 4c 37 8e 20 16 e7 ad 86  64
8e78 : 02 a2 00 9d 00 d8 9d 00  bc
8e80 : d9 9d 00 da 9d 00 db e8  9e
8e88 : d0 f1 60 20 90 98 85 54  f9
8e90 : 4c fb 8b 20 b6 84 20 b8  42
8e98 : 84 20 c5 84 85 b6 20 a0  fe
8ea0 : 85 4c 31 85 20 93 8e a0  62
8ea8 : 00 20 4b 00 45 b6 20 54  be
8eb0 : 00 20 00 82 b0 06 20 03  d2
8eb8 : 85 4c a9 8e 60 20 93 8e  12
8ec0 : a0 00 20 4b 00 18 65 b6  96
8ec8 : 20 54 00 20 00 82 b0 ec  c7
8ed0 : 20 03 85 4c c2 8e 20 3e  fa
8ed8 : 83 c9 45 d0 03 4c a4 8e  ed
8ee0 : c9 41 d0 03 4c bd 8e 4c  64
8ee8 : 45 83 a9 37 85 01 a5 7a  2c
8ef0 : 85 fb a5 7b 85 fc a9 01  34
8ef8 : a8 91 2b 20 33 a5 8a 69  95
8f00 : 02 85 2d a5 23 20 55 a6  9b
8f08 : a5 fc 85 7b a5 fb 85 7a  41
8f10 : 4c 4d 89 20 c2 8b 20 c2  f8
8f18 : 8b 20 c2 8b 4c c2 8b 20  1f
8f20 : c7 8b a9 2c 20 ab 85 20  52
8f28 : f8 80 20 19 8f a0 00 20  ca
8f30 : 4b 00 48 20 e2 80 20 c2  ca
8f38 : 8b 68 20 87 8b a6 b7 f0  9f
8f40 : 0d c8 20 4b 00 20 e2 80  b0
8f48 : 20 c2 8b ca d0 f3 a9 02  5d
8f50 : 38 e5 b7 aa f0 06 20 16  aa
8f58 : 8f ca d0 fa 60 a6 bc bd  89
8f60 : 00 93 48 29 80 08 a9 20  90
8f68 : 28 f0 02 a9 2a 20 ab 85  1b
8f70 : 68 29 7f 20 ab 85 bd 46  bb
8f78 : 93 48 29 80 85 bd 68 29  c4
8f80 : 7f 20 ab 85 bd 8c 93 20  7a
8f88 : 34 90 a5 b7 f0 2e 18 a5  91
8f90 : ba d0 2e a5 bb d0 53 a6  cf
8f98 : b9 bd d2 93 20 ab 85 a4  16
8fa0 : b7 20 4b 00 20 e2 80 88  66
8fa8 : d0 f7 bd da 93 20 ab 85  32
8fb0 : bd e2 93 20 ab 85 bd ea  7b
8fb8 : 93 20 ab 85 a9 0d 4c ab  82
8fc0 : 85 a5 b9 f0 15 a9 28 20  24
8fc8 : ab 85 a4 b7 20 4b 00 20  f2
8fd0 : e2 80 88 d0 f7 a9 29 4c  38
8fd8 : b9 8f a6 aa f0 c1 a9 4d  b6
8fe0 : 20 16 e7 8a 20 e2 80 4c  0a
8fe8 : bc 8f a6 aa d0 f0 a6 b7  09
8ff0 : e8 8a 18 65 fb 85 c1 a5  0e
8ff8 : fc 69 00 85 c2 4c 16 90  62
9000 : a9 37 85 02 a9 0d 86 86  2d
9008 : 02 a9 09 20 73 8e a2 dc  15
9010 : 20 d0 80 4c 55 87 a0 01  58
9018 : 20 4b 00 c9 7f 90 02 c6  29
9020 : c2 18 65 c1 85 c1 90 02  2c
9028 : e6 c2 a5 c2 a4 c1 20 de  c8
9030 : 80 4c bc 8f 20 ab 85 4c  05
9038 : c2 8b 20 a4 84 20 2d 85  65
9040 : a0 00 84 aa 20 1f 8f 20  d0
9048 : 5d 8f a5 bd f0 0d a9 2d  06
9050 : a0 23 20 ab 85 88 d0 fa  d5
9058 : 20 b5 8b 20 00 82 b0 08  21
9060 : 20 6b 90 20 10 85 d0 dc  88
9068 : 4c e3 85 a5 fb 18 65 b7  41
9070 : 85 fb 90 02 e6 fc 4c 03  e5
9078 : 85 a9 00 85 b7 85 b8 85  18
9080 : b9 85 bc 20 2e 83 c9 46  e2
9088 : d0 08 20 17 83 d0 07 e6  f0
9090 : b8 60 c9 0d f0 fb c6 d3  3e
9098 : a0 00 20 4c 20 cf c9 0d  f0 9f
90a0 : f0 c9 41 90 f3 c9 5b b0  34
90a8 : ef 99 00 02 c8 c0 03 d0  e4
90b0 : e9 a2 00 bd 00 93 29 7f  e2
90b8 : cd 00 02 d0 14 bd 46 93  8f
90c0 : 29 7f cd 01 02 d0 0a bd  87
90c8 : 8c 93 29 7f cd 02 02 f0  2f
90d0 : 07 e8 e0 43 d0 dd a2 00  72
90d8 : 86 bc e0 00 f0 b3 20 2e  7e
90e0 : 83 c9 0d d0 04 e6 b8 d0  a1
90e8 : a6 4c 1d 94 c9 30 90 0e  0b
90f0 : c9 47 b0 0a c9 3a 90 04  83
90f8 : c9 41 90 02 38 60 18 60  6e
9100 : 00 19 00 19 05 05 05 05  47
9108 : 00 15 00 00 42 42 42 42  57
9110 : 21 1d 00 1d 05 09 09 09  32
9118 : 00 52 00 52 42 4e 4e 4e  f8
9120 : c2 19 00 19 05 05 05 05  29
```

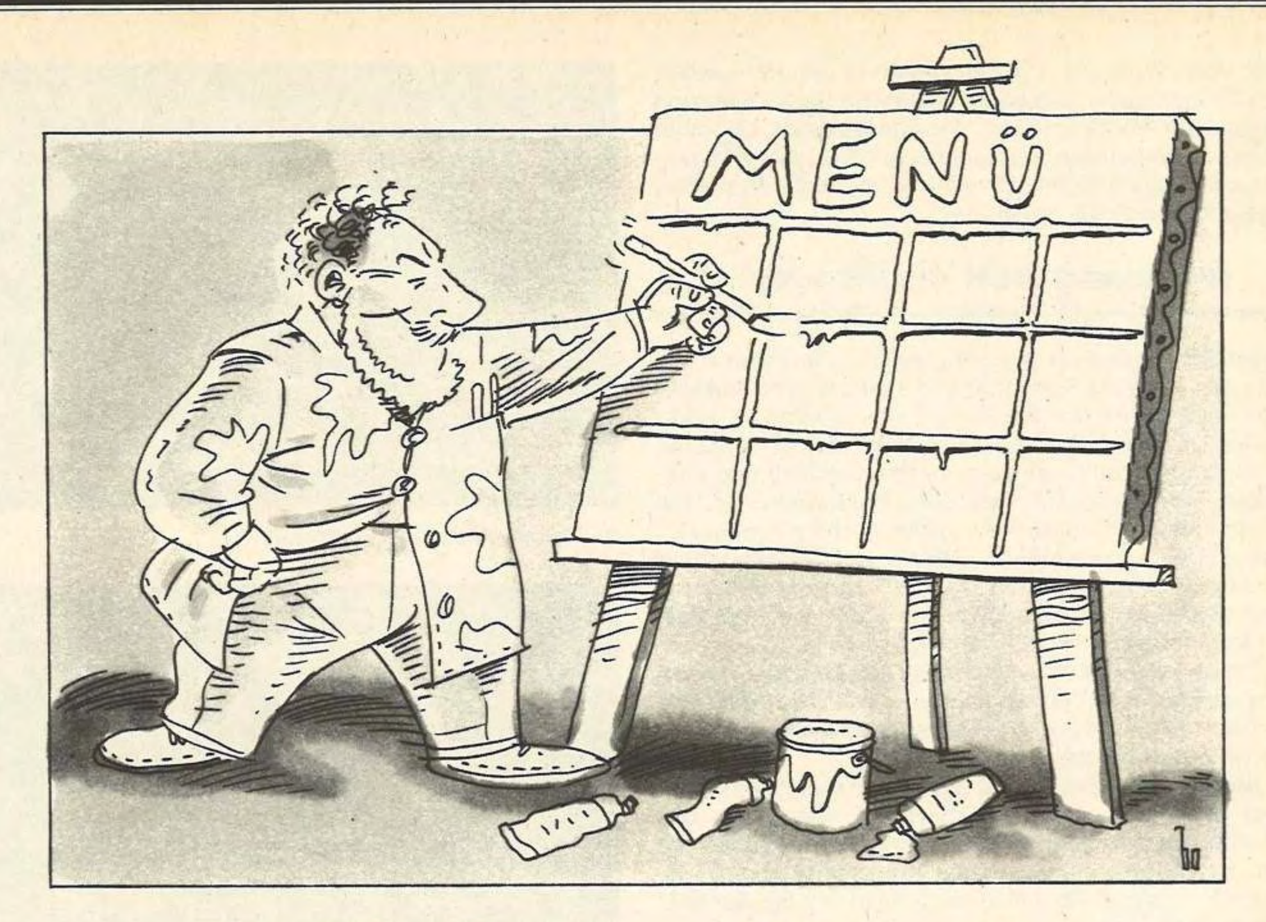
```
9128 : 00 15 00 00 42 42 42 42  77
9130 : 21 1d 00 1d 05 09 09 09  52
9138 : 00 52 00 52 42 4e 4e 4e  18
9140 : 00 19 00 19 05 05 05 05  87
9148 : 00 15 00 00 c2 42 42 42  9f
9150 : 21 1d 00 1d 05 09 09 09  72
9158 : 00 52 00 52 42 4e 4e 4e  38
9160 : 00 19 00 19 05 05 05 05  a7
9168 : 00 15 00 00 9e 42 42 42  7c
9170 : 21 1d 00 1d 05 09 09 09  92
9178 : 00 52 00 52 42 4e 4e 4e  58
9180 : 05 19 05 19 05 05 05 05  0d
9188 : 00 05 00 00 42 42 42 42  cf
9190 : 21 1d 00 00 09 09 11 11  7f
9198 : 00 52 00 00 00 4e 00 00  34
91a0 : 15 19 15 19 05 05 05 05  41
91a8 : 00 15 00 00 42 42 42 42  f7
91b0 : 21 1d 00 1d 09 09 11 11  43
91b8 : 00 52 00 00 4e 4e 52 52  27
91c0 : 15 19 05 19 05 05 05 05  5d
91c8 : 00 15 00 00 42 42 42 42  17
91d0 : 21 1d 00 1d 05 09 09 09  f2
91d8 : 00 52 00 52 42 4e 4e 4e  b8
91e0 : 15 19 05 19 05 05 05 05  7d
91e8 : 00 15 00 00 42 42 42 42  37
91f0 : 21 1d 00 1d 05 09 09 09  12
91f8 : 00 52 00 52 42 4e 4e 4e  d8
9200 : 0b 23 42 3a 39 23 03 3a  a2
9208 : 25 23 00 00 39 23 03 3a  ad
9210 : 0a 23 42 3a 39 23 03 3a  b1
9218 : 0e 23 39 3a 39 23 03 3a  7b
9220 : 1d 02 42 3b 07 02 28 3b  ce
9228 : 27 02 28 00 07 02 28 3b  f2
9230 : 08 02 42 3b 39 02 28 3b  ec
9238 : 2d 02 39 3b 39 02 28 3b  d7
9240 : 2a 18 42 3c 39 18 21 3c  e0
9248 : 24 18 21 00 1c 18 21 3c  40
9250 : 0c 18 42 3c 39 18 21 3c  d2
9258 : 10 18 39 3c 39 18 21 3c  9c
9260 : 2b 01 42 3d 39 01 29 3d  ff
9268 : 26 01 29 00 1c 01 29 3d  42
9270 : 0d 01 42 3d 39 01 29 3d  f1
9278 : 2f 01 39 3d 39 01 29 3d  d9
9280 : 39 30 39 3e 32 30 31 3e  cd
9288 : 17 39 36 00 32 30 31 3e  af
9290 : 04 30 42 00 32 30 31 3e  23
9298 : 38 30 37 00 00 30 00 00  38
92a0 : 20 1e 1f 3f 20 1e 1f 3f  6d
92a8 : 34 1e 33 00 20 1e 1f 3f  a6
92b0 : 05 1e 42 3f 20 1e 1f 3f  2b
92b8 : 11 1e 35 00 20 1e 1f 3f  14
92c0 : 14 12 39 40 14 12 15 40  da
92c8 : 1b 12 16 00 14 12 15 40  19
92d0 : 09 12 42 40 39 12 15 40  74
92d8 : 0f 12 39 40 39 12 15 40  40
92e0 : 13 2c 39 41 13 2c 19 41  f9
92e8 : 1a 2c 22 00 13 2c 19 41  1a
92f0 : 06 2c 42 41 39 2c 19 41  a1
92f8 : 2e 2c 39 41 39 2c 19 41  8f
9300 : 2a 41 41 41 42 42 42 42  07
9308 : 42 42 42 42 42 42 43 43  0e
9310 : 43 43 43 43 43 44 44 44  1e
9318 : 45 49 49 49 4a 4a 4c 4c  3e
9320 : 4c 4c 4e 4f 50 50 50 50  79
9328 : 52 52 52 52 53 53 53 53  46
9330 : 53 53 53 54 54 54 54 54  6e
9338 : 54 d3 d3 d2 d3 d2 d3 cc  82
9340 : c4 c9 c3 c9 00 00 2a 44  0b
9348 : 4e 53 43 43 45 49 4d 4e  ea
9350 : 50 d2 56 56 4c 4c 4c 4c  5b
9358 : 4d 50 50 45 45 45 4f 4e  e2
9360 : 4e 4e cd 53 44 44 44 53  d1
9368 : 4f 52 48 48 4c 4c 4f 4f  fe
9370 : d4 d4 42 45 45 45 54 54  60
9378 : 54 41 41 53 58 58 59 4b  6c
9380 : 4c 4c 52 52 41 41 43 53  a3
9388 : 52 00 00 00 2a 43 44 4c  41
9390 : 43 53 51 54 49 45 4c 4b  e2
9398 : 43 53 43 44 49 56 50 58  17
93a0 : 59 43 58 59 52 43 58 59  2f
93a8 : 50 52 41 58 59 52 50 41  69
93b0 : 41 50 41 50 4c 52 49 53  97
93b8 : 43 43 44 49 41 58 59 58  c4
93c0 : 59 58 41 53 41 50 4f 41  56
93c8 : 45 41 58 58 50 43 41 00  f3
93d0 : 00 00 20 20 20 20 20 23  a6
93d8 : 28 28 20 20 01 01 01 20  7d
93e0 : 2c 29 20 20 2c 2c 2c 20  c2
93e8 : 58 2c 20 20 58 58 59 20  50
93f0 : 29 59 29 58 59 2c 00 00  12
93f8 : 00 00 00 00 00 00 00 00  f9
9400 : 20 67 94 20 ec 90 90 0a  a7
9408 : 20 cf ff 20 ec 90 08 c6  15
9410 : d3 28 08 c6 d3 28 60 20  12
9418 : cf ff 4c 03 94 c9 4d d0  c9
9420 : 20 a5 ab f0 1b 20 00 94  f8
9428 : 90 16 20 c5 84 f0 11 c9  2c
9430 : 11 b0 0d 85 aa 20 cf ff  78
9438 : c9 0d d0 04 a9 03 85 b8  77
9440 : 60 a2 00 86 c1 86 c2 c9  b1
```

```
9448 : 23 d0 24 20 00 94 90 f0  a9
9450 : 20 c5 84 85 c1 20 cf ff  81
9458 : c9 0d d0 e4 a9 15 85 b9  45
9460 : e6 b7 a9 02 85 b8 60 20  ac
9468 : 2e 83 c9 24 f0 f9 60 c9  43
9470 : 28 d0 39 20 00 94 90 f6  28
9478 : 20 c5 84 85 c1 20 2e 83  29
9480 : c9 2c d0 12 a0 02 20 2e  cd
9488 : 83 d9 f1 93 d0 e0 88 d0  be
9490 : f5 a9 19 4c 5e 94 c9 29  2e
9498 : d0 03 4c 21 95 c9 0d f0  df
94a0 : cd c6 d3 20 17 94 90 c6  af
94a8 : 4c 34 95 60 c9 24 d0 03  87
94b0 : 20 2e 83 20 ec 90 b0 01  e4
94b8 : 60 c6 d3 20 00 94 90 f8  4d
94c0 : 20 c5 84 85 c1 20 2e 83  71
94c8 : c9 0d d0 05 a9 05 4c 5e  9d
94d0 : 94 c9 2c d0 15 20 2e 83  80
94d8 : c9 58 f0 09 c9 59 d0 d8  87
94e0 : a9 11 4c 5e 94 a9 09 4c  44
94e8 : 5e 94 c6 d3 20 17 94 90  eb
94f0 : c7 a5 c1 85 c2 20 c5 84  f8
94f8 : 85 c1 20 2e 83 c9 0d d0  88
9500 : 07 a9 42 e6 b7 4c 5e 94  ca
9508 : c9 2c d0 ac 20 2e 83 c9  c6
9510 : 58 f0 09 c9 59 d0 a1 a9  52
9518 : 52 4c 03 95 a9 4e 4c 03  48
9520 : 95 a0 02 20 2e 83 d9 f3  d8
9528 : 93 d0 08 88 d0 f5 a9 1d  d4
9530 : 4c 5e 94 60 a5 c1 85 c2  e0
9538 : 20 c5 84 85 c1 20 2e 83  e9
9540 : c9 29 d0 ef a5 bc c9 1c  6f
9548 : d0 e9 a9 9e 4c 03 95 a2  c3
9550 : ff a5 bc e8 dd 00 92 f0  78
9558 : 09 e0 ff d0 f6 a9 00 85  b3
9560 : b8 60 bd 00 91 a8 29 a0  fc
9568 : d0 0a 98 c5 b9 d0 e2 a9  1d
9570 : 01 85 b8 60 29 80 f0 10  e8
9578 : a5 b9 c9 9e d0 04 a2 6c  d1
9580 : d0 ed c9 42 f0 e9 d0 d5  4f
9588 : a5 b9 c9 42 d0 cf c6 b7  da
9590 : a5 fb 18 69 02 85 c3 a5  0d
9598 : fc 69 00 85 c4 a5 c1 38  ea
95a0 : e5 c3 85 c1 a5 c2 e5 c4  92
95a8 : 85 c2 a5 c1 c9 80 b0 0b  a9
95b0 : a5 c2 d0 0d a9 00 85 c2  c2
95b8 : 4c 6f 95 a5 c2 c9 ff f0  32
95c0 : f3 4c 5d 95 20 34 85 a9  f0
95c8 : 00 85 ab 85 f9 20 b6 84  ab
95d0 : 20 79 90 a5 b8 c9 02 f0  49
95d8 : 0d a9 0d 20 16 e7 c6 d6  6b
95e0 : 20 6c e5 4c 0d 96 20 4f  de
95e8 : 95 a5 b8 f0 ec a0 00 8a  85
95f0 : 20 54 00 a4 b7 f0 09 b9  6a
95f8 : c0 00 20 54 00 88 d0 f7  c2
9600 : a9 00 85 d3 20 1f 8f 20  ff
9608 : 5d 8f 20 6b 90 a0 00 b1  14
9610 : d1 c9 2d d0 05 a9 0d 20  3d
9618 : 16 e7 a9 00 85 d3 a9 2c  82
9620 : 20 16 e7 20 f8 80 a6 fa  6d
9628 : 9a a9 00 85 d3 4c 6f 83  ac
9630 : 20 f6 98 85 fd a5 fc 85  ae
9638 : fe 20 31 85 a9 00 85 f9  e8
9640 : a2 10 9d 32 01 ca 10 fa  35
9648 : 86 ab a9 2c 20 16 e7 20  26
9650 : f8 80 20 c2 8b d0 0a a9  a3
9658 : 0d 20 16 e7 c6 d6 20 6c  74
9660 : e5 20 2e 83 c9 2c d0 28  e3
9668 : a6 d3 e0 01 d0 22 20 00  ef
9670 : 94 90 e4 20 c5 84 85 fc  1a
9678 : 20 17 94 90 da 20 c5 84  2a
9680 : 85 fb 4c 96 96 a9 0d 20  14
9688 : 16 e7 a9 06 85 d3 d0 cc  91
9690 : c6 d3 c9 0d f0 ef 20 2e  bf
9698 : 83 c9 4d d0 20 20 00 94  99
96a0 : 90 e3 20 c5 84 f0 de c9  c1
96a8 : 11 b0 da aa 9d 31 01 0a  99
96b0 : aa a5 fb 9d 10 01 a5 fc  79
96b8 : 9d 11 01 e6 d3 c6 d3 20  fe
96c0 : 79 90 a5 b8 c9 02 f0 0b  88
96c8 : c9 03 f0 49 c9 01 d0 b5  cb
96d0 : 4c 37 97 20 4f 95 a5 b8  4b
96d8 : f0 ab 8a a0 00 20 54 00  a7
96e0 : a4 b7 f0 09 b9 c0 00 20  9f
96e8 : 54 00 88 d0 f7 84 d3 20  ac
96f0 : 1f 8f 20 5d 8f a6 aa f0  45
96f8 : 09 a0 00 84 aa a5 ab 20  a9
9700 : 54 00 20 6b 90 20 00 82  d9
9708 : 90 08 a5 fb 85 ae a5 fc  e3
9710 : 85 af 4c 4a 96 a5 bc a2  98
```

Listing 1. »Promon 64« (Fortsetzung)

```
9718 : 0a dd ff 97 f0 06 ca d0  10
9720 : f8 4c 85 96 bd 09 98 85  04
9728 : ab a9 01 85 b7 a5 aa 85  f7
9730 : c1 bd 13 98 4c db 96 a5  f1
9738 : fd 85 fb a5 fe 85 fc a5  07
9740 : ae 85 fd a5 af 85 fe a9  5b
9748 : 00 85 ab 85 aa 8d 0e dc  af
9750 : a0 00 20 4b 00 a2 0a dd  5b
9758 : 09 98 f0 29 ca d0 f8 aa  7b
9760 : bd 00 91 29 03 85 b7 20  22
9768 : 6b 90 20 00 82 90 e1 ee  36
9770 : 0e dc a5 aa f0 05 a2 15  97
9778 : 20 f0 97 a5 ab f0 05 a2  46
9780 : 00 20 f0 97 60 bd 13 98  31
9788 : 20 54 00 aa bd 00 91 29  9c
9790 : 03 85 b7 c8 20 4b 00 aa  0e
9798 : bd 31 01 d0 1e a5 b7 c9  ca
97a0 : 01 d0 04 a9 fe d0 08 a5  21
97a8 : fb 20 54 00 c8 a5 fc 20  b6
97b0 : 54 00 a9 01 85 ab d0 af  e7
97b8 : 4c 67 97 0a aa bd 10 01  ba
97c0 : 85 c1 bd 11 01 85 c2 a5  4a
97c8 : b7 c9 01 f0 0e a5 c1 20  18
97d0 : 54 00 a5 c2 c8 20 54 00  c5
97d8 : 4c 67 97 20 90 95 d0 06  c7
97e0 : a9 fe 85 aa d0 02 a5 c1  f6
97e8 : 20 54 00 4c 67 97 00 00  ef
97f0 : bd 20 98 f0 06 20 16 e7  8b
97f8 : e8 d0 f5 60 00 00 00 00  d2
9800 : 04 05 06 08 09 0a 0c 0d  35
9808 : 1c 1d 12 22 32 42 52 62  bf
9810 : 72 92 b2 d2 90 b0 f0 30  85
9818 : d0 10 50 70 4c 20 00 00  d8
9820 : 0d 55 4e 44 45 46 49 4e  3c
9828 : 45 44 20 4c 41 42 45 4c  f5
9830 : 28 53 29 0d 00 0d 52 45  2a
9838 : 4c 41 54 49 56 45 20 4f  12
9840 : 55 54 20 4f 46 20 52 41  e3
9848 : 4e 47 45 0d 00 0d 54 52  8b
9850 : 41 43 4b 2f 53 45 43 54  01
9858 : 4f 52 3a 20 20 00 0d 46  26
9860 : 4f 55 4e 44 20 00 0d 50  4d
9868 : 52 45 53 53 20 50 4c 41  d4
9870 : 59 2e 00 ad 02 dd 09 03  cf
9878 : 8d 02 dd ad 00 dd 29 fc  c1
9880 : 05 fb 8d 00 dd 60 a2 12  76
9888 : bd 3e 85 95 4a ca d0 f8  a9
9890 : a9 38 38 e5 02 85 4d 60  62
9898 : 20 31 85 4c b5 8b 00 00  f3
98a0 : 00 00 a2 0d a9 00 18 7d  e1
98a8 : 9b e4 ca d0 fa c9 be d0  1d
98b0 : 03 20 51 f4 4c 00 90 8e  da
98b8 : 11 01 a9 37 85 ae 20 b6  57
98c0 : 84 4c 17 83 20 ed f6 d0  90
98c8 : 02 38 60 4c 00 82 20 f1  00
98d0 : 89 4c 30 8c a2 ff 78 9a  5e
98d8 : d8 e8 8e 16 d0 20 a3 fd  23
98e0 : 20 50 fd 20 15 fd 20 5b  24
98e8 : ff 58 20 53 e4 20 bf e3  9c
98f0 : 20 c9 83 4c a2 98 20 b6  3c
98f8 : 84 85 ae a6 fc 86 af 60  43
9900 : 20 57 f1 c9 0d d0 1c a5  94
9908 : a8 f0 12 48 a2 2d 20 f0  ac
9910 : 97 68 20 e2 80 20 19 8f  cc
9918 : a5 a9 20 e2 80 20 b5 8b  ed
9920 : 4c e3 85 c9 4e d0 12 ad  07
9928 : 00 cf f0 0a 85 a8 ad 01  e4
9930 : cf 85 a9 4c 3d 99 4c 45  12
9938 : 83 c9 52 d0 54 20 ae 99  83
9940 : a9 01 8d a7 00 20 fd 99  ee
9948 : a2 0d 20 0e f2 a0 00 20  af
9950 : 57 f1 99 00 cf c8 d0 f7  7d
9958 : 20 a0 9a a9 4d 4c 95 99  65
9960 : 20 ae 99 a9 02 8d a7 00  9e
9968 : 20 66 9a a2 0d 20 50 f2  af
9970 : a0 00 b9 00 cf 20 ca f1  8c
9978 : a6 90 d0 03 c8 d0 f3 20  1e
9980 : 33 f3 a9 32 20 fd 99 a9  09
9988 : 0d 20 91 f2 a9 0f 4c 91  cf
9990 : f2 c9 57 f0 cb aa a0 0a  03
9998 : 84 c6 b9 94 9a 99 76 02  d5
99a0 : 88 d0 f7 8e 77 02 a9 91  b1
99a8 : 20 ba 8b 4c 6f 83 20 17  53
99b0 : 83 f0 0f 20 c5 84 f0 42  3c
99b8 : 85 a8 20 c5 84 85 a9 4c  06
99c0 : cb 99 ad a7 00 f0 33 a5  58
99c8 : a8 f0 2f a0 0d b9 7f 9a  9a
99d0 : 99 11 01 88 d0 f7 a9 0f  d5
99d8 : a8 a6 f8 20 74 85 a9 00  30
99e0 : 20 f9 fd 20 4a f3 a9 0d  85
99e8 : a8 a6 f8 20 00 fe a9 01  c6
99f0 : a2 9f a0 9a 20 f9 fd 4c  40
99f8 : 4a f3 4c 45 83 8d 13 01  eb
9a00 : a5 a8 20 59 9a 8e 1a 01  b5
9a08 : 8d 1b 01 a5 a9 20 59 9a  4e
9a10 : 8e 1d 01 8d 1e 01 a2 0f  b1
9a18 : 20 50 f2 a2 00 bd 12 01  a9
9a20 : 20 ca f1 e8 e0 0d 90 f5  e3
9a28 : 20 33 f3 a9 00 85 90 20  c3
9a30 : 1f 83 a5 f8 20 09 ed a9  ef
9a38 : 6f 20 c7 ed 20 13 ee c9  51
9a40 : 30 d0 06 4c ef ed 20 13  f8
9a48 : ee 20 ca f1 c9 0d d0 f6  6d
9a50 : 20 ef ed 20 87 99 4c 4a  f2
9a58 : 83 a2 30 38 e9 0a 90 03  77
9a60 : e8 b0 f9 69 3a 60 a2 0f  9b
9a68 : 20 50 f2 a2 00 bd 8d 9a  1b
9a70 : 20 ca f1 e8 e0 08 90 f5  0b
9a78 : 4c 33 f3 00 00 00 00 00  5b
9a80 : 55 01 3a 31 33 20 30 20  40
9a88 : 31 38 20 30 30 42 2d 50  4e
9a90 : 20 31 33 20 30 cc 43 46  1d
9a98 : 30 30 43 46 46 46 0d 23  8b
9aa0 : 20 cc ff 4c 87 99 a2 c1  03
9aa8 : 20 b8 84 20 b8 84 a2 ae  e1
9ab0 : 20 b8 84 20 b6 84 20 b8  d3
9ab8 : 84 a9 00 85 f9 8d 0e dc  c0
9ac0 : 4c 34 85 a0 00 20 4b 00  ca
9ac8 : 20 87 8b a5 b8 f0 3b c8  d5
9ad0 : 20 4b 00 85 aa c8 20 4b  4f
9ad8 : 00 85 ab a5 aa 38 e5 c1  c2
9ae0 : aa a5 ab e5 c2 90 23 a8  93
9ae8 : a5 aa c5 c3 a5 ab e5 c4  a5
9af0 : b0 18 8a 18 65 ae 85 aa  89
9af8 : 98 65 af 85 ab a0 01 a5  ee
9b00 : aa 20 54 00 c8 a5 ab 20  78
9b08 : 54 00 20 00 82 b0 06 20  6a
9b10 : 6b 90 4c c3 9a ee 0e dc  62
9b18 : 60 20 a6 9a 4c c3 9a 20  13
9b20 : a6 9a a2 04 b5 fa 9d 15  10
9b28 : 01 b5 c0 9d 11 01 95 fa  4d
9b30 : b5 ad 9d 19 01 95 c0 ca  9c
9b38 : d0 ea 20 f0 8a a2 04 bd  ed
9b40 : 11 01 95 c0 bd 15 01 95  03
9b48 : fa ca d0 f3 ad 1a 01 85  15
9b50 : ae ad 1b 01 85 af ce 0e  e9
9b58 : dc 20 34 85 4c c3 9a 20  90
9b60 : b6 84 20 b8 84 a0 00 20  05
9b68 : 17 83 f0 09 20 c5 84 99  14
9b70 : 12 01 c8 d0 f2 84 b6 c0  ff
9b78 : 00 d0 03 4c 45 83 20 98  4d
9b80 : 98 a0 00 20 4b 00 d9 12  ad
9b88 : 01 d0 0b c8 c4 b6 d0 f3  fa
9b90 : 20 f8 80 20 c2 8b 20 c4  e3
9b98 : 98 b0 06 20 03 85 4c 81  9f
9ba0 : 9b 20 b5 8b 4c e3 85 ad  7f
9ba8 : 11 d0 09 20 8d 11 d0 ad  68
9bb0 : 16 d0 0d e4 81 8d 16 d0  8d
9bb8 : a9 55 85 fc 20 7d 8e 20  75
9bc0 : e4 ff c9 20 d0 27 [illegible] 03  0d
9bc8 : 85 fb 20 73 98 a9 15 8d  08
9bd0 : 18 d0 ad 11 d0 29 df 8d  cf
9bd8 : 11 d0 ad 16 d0 29 ef 8d  b1
9be0 : 16 d0 a2 00 8e 20 d0 8e  51
9be8 : 21 d0 4c 76 8e c9 85 d0  42
9bf0 : 03 ee 20 d0 c9 86 d0 03  a7
9bf8 : ee 21 d0 c9 87 d0 0a e6  d9
9c00 : fc a5 fc 20 7d 8e 4c bf  0f
9c08 : 9b c9 48 d0 0b ad 16 d0  cc
9c10 : 29 ef 8d 16 d0 4c bf 9b  fd
9c18 : c9 4d d0 0b ad 16 d0 09  fe
9c20 : 10 8d 16 d0 4c bf 9b c9  5b
9c28 : 30 90 94 c9 39 b0 90 38  ca
9c30 : e9 31 4a 85 fb 90 07 ad  b1
9c38 : 18 d0 09 08 d0 05 ad 18  18
9c40 : d0 29 f7 8d 18 d0 a9 03  09
9c48 : 38 e5 fb 85 fb 20 73 98  e2
9c50 : 4c bf 9b a0 00 20 4b 00  a5
9c58 : 48 20 03 85 20 4b 00 85  89
9c60 : fc 68 85 fb 60 68 68 a9  af
9c68 : 1d 48 a9 87 48 ba ea ba  80
9c70 : 20 b7 98 f0 0d 20 c5 84  a2
9c78 : 85 ae 20 17 83 f0 03 4c  a4
9c80 : 45 83 a2 0e bd 8e 9c 95  df
9c88 : 4a ca d0 f8 4c 9d 9c 78  a0
9c90 : a5 ae 85 01 b1 fb 48 a9  7d
9c98 : 37 85 01 68 60 a9 0d 20  a7
9ca0 : 16 e7 20 16 e7 a5 ae 20  1b
9ca8 : e2 80 20 19 8f a2 00 bd  7f
9cb0 : 0d 01 20 e2 80 20 c2 8b  cd
9cb8 : e8 e0 05 d0 f2 a2 2d bd  e0
9cc0 : cc 9c 9d 11 01 ca d0 f7  fd
9cc8 : 86 b6 4c fa 9c 85 01 ad  71
9cd0 : 0d 01 48 ad 0e 01 28 ea  85
9cd8 : ea ea 08 78 8d 0e 01 8e  b3
9ce0 : 0f 01 8c 10 01 68 8d 0d  39
9ce8 : 01 a5 01 85 ae a9 37 85  cd
9cf0 : 01 4c 2d 9e 08 e6 b6 4c  62
9cf8 : 20 01 20 c2 8b a9 2e 20  f8
9d00 : ab 85 20 f8 80 20 c2 8b  c0
9d08 : a0 00 20 4b 00 8d 1c 01  f9
9d10 : 20 87 8b 20 5d 8f ad 14  0c
9d18 : 03 85 fd ad 15 03 85 fe  90
9d20 : a9 31 8d 14 03 a9 ea 8d  8c
9d28 : 15 03 58 a5 91 c9 7f d0  91
9d30 : 0b 20 e3 85 ad 11 01 85  67
9d38 : fa 4c 4a 83 20 3e f1 ae  74
9d40 : 1c 01 c9 20 f0 0b c9 4a  76
9d48 : d0 e1 e0 20 d0 dd 4c 07  80
9d50 : 9e a5 ba f0 37 e0 4c d0  db
9d58 : 18 20 03 85 20 53 9c 78  f2
9d60 : a5 fd 8d 14 03 a5 fe 8d  5e
9d68 : 15 03 ba 8e 11 01 4c a5  15
9d70 : 9c e0 20 d0 0e a5 fb 18  cd
9d78 : 69 02 48 a5 fc 69 00 48  55
9d80 : 4c 59 9d 20 03 85 20 53  68
9d88 : 9c 4c 5c 9d e0 60 d0 0c  81
9d90 : 68 85 fc 68 85 fb 20 03  c6
9d98 : 85 4c 5f 9d e0 40 d0 07  30
9da0 : 68 8d 0d 01 4c 90 9d e0  b4
9da8 : 00 d0 58 a5 fb 18 69 02  05
9db0 : 48 a5 fc 69 00 48 ad 0d  4a
9db8 : 01 48 a9 fe 85 fb a9 ff  06
9dc0 : 85 fc a0 00 20 4b 00 aa  9d
9dc8 : c8 20 4b 00 a8 a5 ae 29  38
9dd0 : 02 d0 08 ea 85 fc 86 fb  ec
9dd8 : 4c 5f 9d ad 16 03 c9 66  5e
9de0 : f0 18 c9 1f d0 ee ad 17  9c
9de8 : 03 c9 87 d0 e7 20 e3 85  e6
9df0 : ad 11 01 85 fa a9 52 4c  f6
9df8 : 7e 83 ad 17 03 c9 fe d0  a2
9e00 : d3 f0 ea 4c 57 9e ea a4  ef
9e08 : b7 f0 09 20 4b 00 99 1c  d1
9e10 : 01 88 d0 f7 ae 11 01 9a  35
9e18 : a5 fd 8d 14 03 a5 fe 8d  16
9e20 : 15 03 ae 0f 01 ac 10 01  fc
9e28 : a5 ae 4c 12 01 ba 8e 11  bc
9e30 : 01 20 6b 90 4c a5 9c a9  e6
9e38 : 1b 8d 1d 01 a9 44 8d 37  e3
9e40 : 01 4c 14 9e ba 8e 11 01  a7
9e48 : a5 b6 f0 e5 a5 c1 85 fb  b8
9e50 : a5 c2 85 fc 4c a5 9c a5  07
9e58 : bb d0 dc a5 bc c9 42 f0  6c
9e60 : 03 4c 07 9e a9 6e a0 9e  ed
9e68 : 20 1e ab 4c 31 9d 0d 43  c6
9e70 : 52 41 20 42 4c 4f 43 4b  96
9e78 : 53 20 43 50 55 0d 00 ea  4a
9e80 : ea ea a9 01 85 b9 20 17  3f
9e88 : 83 f0 07 c6 b9 a2 c3 20  1e
9e90 : b8 84 20 a8 9e a5 90 d0  a2
9e98 : 0c a2 10 20 d0 80 a4 c3  29
9ea0 : a5 c4 4c de 80 4c 4a 83  31
9ea8 : 20 32 9f a5 ab c9 02 f0  71
9eb0 : 04 c9 01 d0 f3 a9 00 8d  9b
9eb8 : 51 03 a9 5e a0 98 20 1e  4c
9ec0 : ab a9 41 a0 03 20 1e ab  a5
9ec8 : 58 a5 91 c9 7f f0 d6 18  9b
9ed0 : a5 c5 c9 3c d0 f3 78 a5  2c
9ed8 : b9 f0 0a ad 3c 03 85 c3  bb
9ee0 : ad 3d 03 85 c4 a0 00 84  f8
9ee8 : 90 ad 3e 03 38 ed 3c 03  29
9ef0 : 08 18 65 c3 85 ae ad 3f  d9
9ef8 : 03 65 c4 28 ed 3d 03 85  c4
9f00 : af a2 00 20 d0 80 a5 c4  36
9f08 : a4 c3 20 de 80 20 47 9f  d7
9f10 : a5 bd 45 d7 05 90 f0 04  81
9f18 : a9 ff 85 90 18 60 a0 00  3b
9f20 : 84 c0 ad 11 d0 29 ef 8d  c3
9f28 : 11 d0 ca d0 fd 88 d0 fa  cb
9f30 : 78 60 20 99 9f c9 00 f0  3e
9f38 : f9 85 ab 20 c7 9f 91 b2  08
9f40 : c8 c0 c0 d0 f6 f0 23 20  76
9f48 : 99 9f 20 c7 9f 91 c3 18  78
9f50 : 45 d7 85 d7 e6 c3 d0 02  b1
9f58 : e6 c4 a5 c3 c5 ae a5 c4  74
9f60 : e5 af 90 e6 20 c7 9f 20  1d
9f68 : 1e 9f c8 84 c0 58 18 a9  9b
9f70 : 00 8d a0 02 08 78 ad 11  bc
9f78 : d0 09 10 8d 11 d0 20 ca  30
9f80 : fc a9 7f 8d 0d dc 20 dd  d6
9f88 : fd ad a0 02 f0 09 8d 15  7c
9f90 : 03 ad 9f 02 8d 14 03 28  68
9f98 : 60 20 ee 9f 20 1e 9f 84  32
9fa0 : d7 a9 07 8d 06 dd a2 01  9b
9fa8 : 20 d7 9f 26 bd a5 bd c9  f4
9fb0 : 02 d0 f5 a0 09 20 c7 9f  9c
9fb8 : c9 02 f0 f9 c4 bd d0 e8  4d
9fc0 : 20 c7 9f 88 d0 f6 60 a9  56
9fc8 : 08 85 a3 20 d7 9f 26 bd  0e
9fd0 : c6 a3 d0 f7 a5 bd 60 a9  b8
9fd8 : 10 2c 0d dc f0 fb ad 0d  9d
9fe0 : dd 8e 07 dd 48 a9 19 8d  d3
9fe8 : 0f dd 68 4a 4a 60 20 22  b6
9ff0 : 84 f0 08 a2 46 20 f0 97  9b
9ff8 : 20 22 84 d0 fb 60 00 00  27
```

Listing 1. »Promon 64« (Schluß)



Pull-down-Menüs in Maschinensprache

Wollen Sie Ihren Programmen ein professionelles Aussehen geben? Dann arbeiten Sie doch in Zukunft mit Pull-down-Menüs. Wie das funktioniert und wie Sie die Pull-down-Menüs in Ihre Basic- oder Maschinenprogramme einbinden können, erkären wir hier anhand eines dokumentierten Quell-Listings.

Die meisten unter Ihnen wissen bereits, was unter einem »Pull-down-Menü« zu verstehen ist. Eine Vielzahl moderner Programme arbeitet mit dieser interessanten Menüform, vorwiegend auf grafikorientierten Computern wie dem Atari ST oder dem Commodore Amiga.

Bild 1 zeigt ein Beispiel eines Pull-down-Menüs. Es besteht aus einer »Menüleiste«, die die Namen verschiedener »Untermenüs« enthält und sich meist in der obersten Bildschirmzeile befindet.

Das aktuelle oder auch »aktive« Untermenü wird in einem Fenster oder »Window« unterhalb des Menünamens angezeigt. Mit den Cursor-Tasten (oder einer Maus) wird das gewünschte Kommando ausgewählt. Der Benutzer bewegt mit <CRSR>-unten/-oben einen vergrößerten Cursor (nicht nur ein Zeichen, sondern das gesamte Kommando wird invertiert) zum gewünschten Befehl und bestätigt die Auswahl mit <RETURN>.

Mit den Tasten <CRSR>-rechts/-links wird das momentan aktive Untermenü wieder »weggeklappt« und das rechte beziehungsweise linke Untermenü aktiviert (Bild 2).

Diese Art der Menüverwaltung ist extrem komfortabel, da der Benutzer mit den Cursor-Tasten in beliebigen Untermenüs »herumblättern« und sich auf diese Weise geradezu spielerisch aus einer Vielzahl gebotener Kommandos das gewünschte auswählen kann.

Leider besitzt der C 64 einen gravierenden Nachteil im Vergleich zum Atari ST oder auch zum Amiga. Im Gegensatz zum C 64 unterstützen die Betriebssysteme beider Computer diese Menüform. Pull-down-Menüs sind gewissermaßen »serienmäßig« eingebaut.

Dieser Artikel richtet sich vorwiegend an Leser, die bereits gute Grundkenntnisse in der Assembler-Programmierung des C 64 besitzen.

Ziel ist es, jedem C 64-Besitzer die benötigten Hilfsmittel zur Verwaltung von Pull-down-Menüs zu verschaffen. Diese Hilfsmittel bestehen aus mehreren Assembler-Routinen, die Sie – wenn Sie nicht über Assembler-Kenntnisse verfügen – abtippen und mit der Kurzanleitung am Ende des Kapitels problemlos in Ihren eigenen Basic-Programmen anwenden können.

Assembler-Programmierer können jedoch erheblich größeren Nutzen aus diesem Artikel ziehen:

1. Wenn Sie sich noch nicht zu den Profis zählen, wird Ihnen dieser Artikel sicher einiges über die Planung und Vorgehensweise bei der Erstellung größerer Programmpakete vermitteln, zum Beispiel über die wichtige modulare Programmierung.

2. Das gesamte Programmpaket besteht aus einzelnen Routinen, die in sich abgeschlossen sind und unabhängig

von der Verwaltung von Pull-down-Menüs genutzt werden können. Schnittstellen existieren sowohl für Basic- wie auch für Assembler-Programmierer. Wie Sie einzelne Routinen über diese Schnittstellen ansprechen, erfahren Sie im folgenden Hauptteil des Artikels, der die Planung und den Aufbau des Gesamtprogramms behandelt.

Am Anfang steht die Planung

Im folgenden werden wir das Programm »gemeinsam« entwickeln. Die geplante Funktionsweise ist bekannt. Mit den Cursor-Tasten soll ein inverser Balken – der »Cursor« – innerhalb eines Untermenüs bewegt und damit der gewünschte Untermenüpunkt selektiert werden. Ebenfalls mit den Cursor-Tasten wird bestimmt, welches Menü aktiv ist. Mit <CRSR>-rechts/-links wird das aktive Menü gewechselt.

Diese bei Pull-down-Menüs übliche Art der Steuerung besitzt einen entscheidenden Nachteil. Angenommen, Sie arbeiten täglich mit einem bestimmten Programm, das Pull-down-Menüs wie in Bild 1 beziehungsweise Bild 2 verwendet. Nach kurzer Zeit werden Ihnen die verschiedenen Kommandos vertraut sein. Wahrscheinlich werden Sie es nun als umständlich empfinden, zum Beispiel zum Kopieren der Diskette fünf Tasten betätigen zu müssen.

Bei dem in den Bildern verwendeten Pull-down-Menü drücken Sie zuerst die Taste <CRSR>-rechts, um das Untermenü »Disk« zu aktivieren. Anschließend drücken Sie viermal <CRSR>-unten, um das Kommando »Copy« zu selektieren, und zum Abschluß bestätigen Sie die Auswahl mit <RETURN>.

Ein wirklich komfortables Verwaltungsprogramm sollte daher außer der Auswahl mit den Cursor-Tasten auch eine »Direktanwahl« des gewünschten Kommandos zulassen. Mein Vorschlag, der im vorgestellten Programm verwirklicht wurde, sieht so aus:

1. Ein Menü kann mit Hilfe des Anfangsbuchstabens aktiviert werden. Gleichzeitig muß die Taste <CTRL> gedrückt werden. Die Kombination <CTRL+D> aktiviert zum Beispiel das Untermenü »Disk«.

2. In einem aktivierten Untermenü kann ein Kommando direkt angewählt werden, indem der Anfangsbuchstabe des Kommandonamens eingegeben wird. Beispiel: Wenn das Untermenü »Disk« aktiviert ist, genügt die Betätigung der Taste <C>, um das Kommando »Copy« anzuwählen.

Mit dieser Methode kann jedes Kommando durch zwei Tasten angewählt werden. Voraussetzung ist natürlich, daß innerhalb eines Menüs jedes Kommando mit einem anderen Anfangsbuchstaben beginnt. Gleiches gilt für die Menüleiste: Die Anfangsbuchstaben aller Untermenünamen müssen eindeutig sein, ein Anfangsbuchstabe darf keinesfalls mehrfach vorkommen.

Die Frage ist nun, welche Routinen benötigt werden, um ein solches Pull-down-Menü zu verwalten:

1. Eine Routine zur Erzeugung des »Cursors«, die ein Menükommando invertiert. Diese Routine soll zusätzlich verwendet werden, um in der Menüleiste den Namen des momentan aktiven Untermenüs zu invertieren. Um die Routine möglichst vielseitig – auch außerhalb des vorgestellten Programms – verwenden zu können, wird eine allgemeine »Invertier-/Normalisier-Routine« erstellt, die beliebige Bildschirmausschnitte invertiert beziehungsweise wieder normalisiert.

2. Eine »Mal-Routine«, die mit den Grafikzeichen des C 64 einen Rahmen zeichnet, in dem das aktuelle Untermenü dargestellt wird. Diese Routine soll zusätzlich die verschiedenen Kommandos in diesen Rahmen schreiben.

3. Eine »Window-Routine«, mit der es möglich ist, vor dem Zeichnen eines Untermenüs den aktuellen Bildschirminhalt zu retten, bevor ihn das »Herunterklappen« des Menüs überschreibt. Wird das Untermenü wieder »zugeklappt«, soll die gleiche Routine den geretteten Bildschirminhalt zurückschreiben.

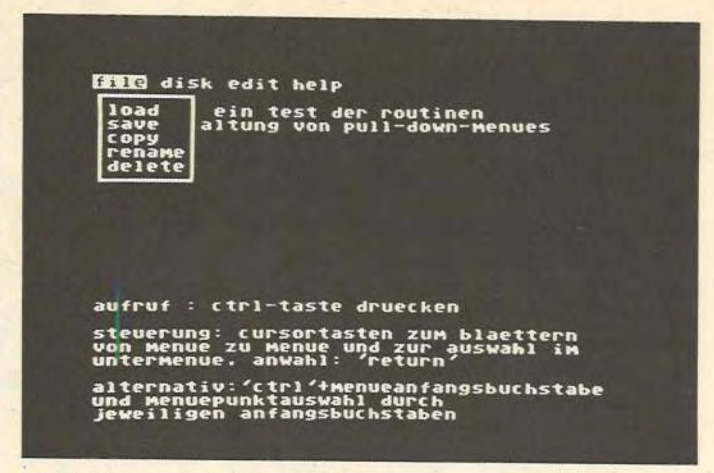


Bild 1. Untermenü »File« aktiviert



Bild 2. Untermenü »Disk« aktiv Kommando »Copy« angewählt

Von allen drei Routinen ist die Window-Routine am vielseitigsten verwendbar, wenn sie allgemein genug erstellt wird. Mit dieser Routine ist es jederzeit möglich, ein Pull-down-Menü zum Beispiel über einen Text oder Datensatz zu legen und nach Auswahl eines Kommandos den ursprünglichen Zustand des Bildschirms wiederherzustellen.

Ein weiteres Beispiel für die Anwendung dieser Routine sind Hilfstexte, die – dank der Mal-Routine mit einem netten Rahmen umgeben – jederzeit eingeblendet werden können. Wenn der Benutzer den Hilfstext nicht mehr benötigt, wird ebenfalls der alte Bildschirminhalt zurückgeschrieben.

Um eine wirklich absolut professionelle Bildschirmgestaltung zu ermöglichen, sollte die Window-Routine allgemein genug sein, um mehrere (!) Windows überlagern zu können (Bild 3).

Diese – für Pull-down-Menüs nicht benötigte – zusätzliche Routine ermöglicht Bildschirmgestaltungen wie auf dem Atari ST oder dem Amiga. Der Benutzer ruft ein Window mit einem Hilfstext auf und überlagert den Hilfstext mit einem weiteren Window, das zum Beispiel eine ASCII-Tabelle oder das Directory enthält. Mit einem Tastendruck wird das oberste Window »entfernt« und das Window mit dem Hilfstext erscheint wieder vollständig. Mit einem weiteren Tastendruck wird der Hilfstext ausgeblendet und der ursprüngliche Bildschirminhalt ist wiederhergestellt.

Welche fantastischen Programme mit dieser Window-Routine erstellt werden können, liegt allein an Ihnen und Ihrer Fantasie.

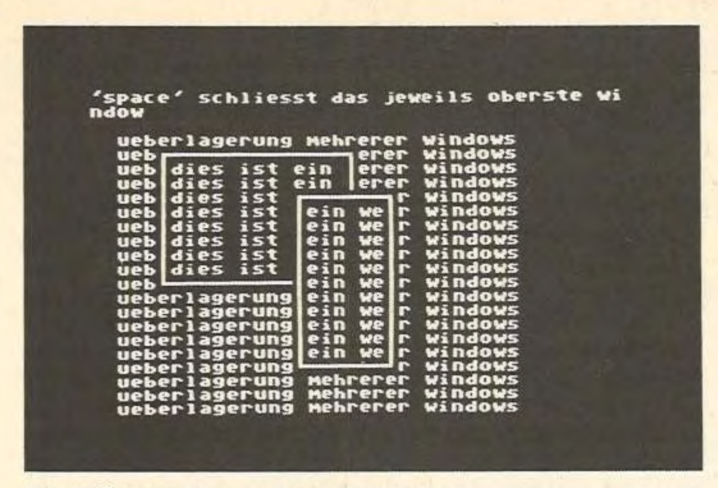


Bild 3. Überlagerung mehrerer Windows

Doch zurück zu unserem Programmpaket. Die wichtigsten Routinen sind festgelegt und werden im folgenden beschrieben. Ein Problem, das für alle Routinen gilt, ist die Art und Weise der Parameterübergabe. Alle Routinen sollen sowohl von Basic als auch von Maschinensprache aus aufzurufen sein.

Die Parameterübergabe wird definiert

Die Parameterübergabe von Basic stellt die größeren Probleme. Es muß vorausgesetzt werden, daß Sie – zum Beispiel aus einem Artikel in der 64'er – wissen, wie ein Basic-Programm Parameter an ein Maschinenprogramm übergibt und wie der Basic-Interpreter Variablen verwaltet (Wenn nicht: In den Sonderheften 1/86, 7/86, 8/86 und 9/86 finden Sie verschiedene Artikel, die die erforderlichen Routinen beschreiben).

Zusammenfassung der Routinen zur Parameterübergabe:

CHKKOM ($AEFD): Liest ein Zeichen aus dem Basic-Text und prüft, ob es sich um ein Komma handelt.

GETBYT ($B79E): Liest einen Ein-Byte-Wert (Angabe von Variablen möglich) aus dem Basic-Text und übergibt ihn im X-Register.

GETPOS ($B08B): Liest eine beliebige Variable ein und übergibt in Akku und Y-Register (Y=Low-, Akku=High-Byte) einen Zeiger auf die Variable (bei numerischen Variablen) beziehungsweise auf die Stringdescriptoren (wenn eine Stringvariable übergeben wird).

Mit diesen Routinen können wir alle Parameter einlesen, die ein Aufruf wie zum Beispiel SYS 49152,10,20,X,A$(10) an ein Maschinenprogramm übergibt.

Um Sie nicht in heillose Verwirrung zu stürzen, benutzen alle Routinen sehr einheitliche Parameter. Sowohl in der Invertier- als auch in der Mal- und der Window-Routine wird jeweils ein Rechteck behandelt (invertiert, gemalt oder »gerettet«). Die Rechteck-Parameter sollten daher allen Routinen in einheitlicher Art und Weise mitgeteilt werden. Im vorgestellten Programm wird allen Routinen die obere linke Ecke (Spalte und Zeile), die Breite und die Länge des Rechtecks übergeben. Das betreffende Rechteck ist durch diese Parameter eindeutig definiert. Wichtig: Bildschirmkoordinaten wie Spalte und Zeile werden ab Null numeriert (Spalte 0 bis 39; Zeile 0 bis 24).

Der Aufruf der einzelnen Routinen wird lauten:

1. Invertier-Routine: SYS xxxx,spalte,zeile,breite,länge,flag.

Die Parameter »spalte, zeile, breite, länge« definieren das Rechteck, »flag« entscheidet darüber, ob der Bildschirmausschnitt invertiert oder normalisiert werden soll (flag=0 => normalisieren; flag=1 => invertieren). Beispiel: SYS xxxx, 5,10,15,20,1 invertiert ein Rechteck mit der linken oberen Ecke 5/10 (Spalte/Zeile), das 15 Spalten breit und 20 Zeilen lang ist.

2. Mal-Routine: SYS xxxx,spalte,zeile,breite,länge,array.

Wie erläutert, füllt die Mal-Routine einen Rahmen mit den Kommandos eines Untermenüs. Diese Kommandos übergibt das Basic-Programm in einem Stringarray. Das in Bild 1 dargestellte Untermenü kann wie folgt gezeichnet werden: SYS xxxx,0,1,8,7,A$(2). Voraussetzung: Das Array A$(...) muß folgende Strings enthalten:

```
...
A$(2)="LOAD"
A$(3)="SAVE"
A$(4)="COPY"
A$(5)="RENAME"
A$(6)="DELETE"
...
```

3. Window-Routine: SYS xxxx,spalte,zeile,breite,länge,flag, puffer.

Der Parameter »flag« entscheidet darüber, ob ein Bildschirmausschnitt in einen Pufferbereich des Computerspeichers kopiert oder aus diesem Bereich auf den Bildschirm zurückgeschrieben wird (flag=0 => Puffer nach Screen übertragen; flag=1 => Screen nach Puffer übertragen). Da die Überlagerung mehrerer Windows möglich sein soll, werden mehrere Puffer verwendet, um ein Überschreiben des Pufferinhalts durch ein zusätzliches Window zu verhindern. Mit »puffer« wird die Nummer des gewünschten Puffers angegeben (die Numerierung beginnt mit Puffer Nummer Null).

Der Programmablauf wird entwickelt

Nachdem nun alle Anforderungen an die einzelnen Routinen besprochen wurden, wird im folgenden Abschnitt das Programm anhand des Source-Codes erläutert (Listing 1). Das Programm wurde mit dem Hypra-Ass entwickelt, der Ihnen sicherlich aus dem Assembler-Sonderheft 8/85 und dem 64'er-Stammheft (Ausgabe 7/85) bekannt ist.

Verwendete Label

Den ersten Abschnitt bildet wie üblich die Definition verschiedener Label (Bild 4). Die verwendeten Interpreter-Routinen CHKKOM, GETBYT und GETPOS wurden kurz erläutert, die Betriebssystem-Routinen GETIN (Zeichen von der Tastatur einlesen), BSOUT (Zeichen ausgeben) und PLOT (Cursor setzen/aktuelle Position holen) werden als bekannt vorausgesetzt.

Interessanter sind die »programminternen« Label. In PARBACK und PARBACK+1 übergibt das Hauptprogramm zur Verwaltung der Pull-down-Menüs die Nummer des selektierten Untermenüs und die Nummer des darin angewählten Kommandos an das aufrufende Programm zurück.

INDIZ und INDIZ+1 sind je zwei Speicherzellen der Zero-Page, die für indirekt indizierte Adressierung verwendet werden (LDA (INDIZ),Y). In STRLEN wird die Länge und in STRPOS beziehungsweise STRPOS+1 die Adresse eines Strings abgelegt, das heißt in diese drei Speicherzellen werden die Inhalte der drei Stringdescriptoren kopiert (Länge, Pointer low, Pointer high). CNTI und CNTI+ werden für verschiedene Zwecke als Zähler eingesetzt (Schleifen etc.).

COL, LINE, BREITE und LAENGE werden zur Speicherung der erläuterten Rechteck-Parameter verwendet (spalte, zeile, breite, länge). In FLAG und PUFNR werden die zusätzlichen Parameter »flag« und »puffer« abgelegt. Das Label ROUTIN erhält erst in einem späteren Abschnitt des Source-Codes eine Bedeutung, wo ab der Adresse dieses Labels weitere Parameter abgelegt werden.

Für die Window-Routine werden einige zusätzliche Label benötigt. SCREENP und PUFFERP nehmen jeweils einen

```
10   -;***************************
20   -;* routinen zur verwaltung *
30   -;*  von pull-down-menues   *
40   -;*   (c) s.baloui, 1986    *
50   -;***************************
60   -;
70   -;
80   -;
100  -;*** label: betriebssystem ***
110  -.eq getin    = $ffe4  ;zeichen von tastatur lesen
120  -.eq chkkom   = $aefd  ;basic-text: naechstes zeichen: komma?
130  -.eq getbyt   = $b79e  ;basic-text: bytewert holen (x-register)
140  -.eq getpos   = $b08b  ;basic-text: pointer auf variable holen
150  -.eq plot     = $fff0  ;cursor setzen
160  -.eq bsout    = $ffd2  ;zeichen aussgeben
180  -;
182  -;
183  -;*** label: programmintern ***
185  -.eq parback  = $a7       ;auswahlrueckuebergabe an basic
190  -.eq indiz    = $a9       ;pointer f.indizierte adressierung
200  -.eq indiz1   = $ab       ;pointer f.indizierte adressierung
202  -.eq strlen   = $ad       ;laenge der von basic uebergeb.stringvariablen
204  -.eq strpos   = $ae       ;pointer auf uebergeb.stringvariable
210  -.eq cnti     = $0334     ;zaehler
220  -.eq cntj     = cnti+1    ;zaehler
230  -.eq col      = cntj+1    ;spalte merken
240  -.eq line     = col+1     ;zeile merken
250  -.eq breite   = line+1    ;rechteckbreite merken
260  -.eq laenge   = breite+1 ;rechtecklaenge merken
270  -.eq flag     = laenge+1
280  -.eq pufnr    = flag+1    ;nr.des anzusprechenden puffers
310  -.eq routin   = pufnr+2
320  -;
330  -;
340  -;
350  -;*** labels f.puffer-routine ***
360  -.eq screenp  = indiz  ;pointer auf bildschirm
370  -.eq pufferp  = indiz1 ;pointer auf puffer
380  -.eq pufpoi   = $0293  ;tabelle mit startadressen der aktuellen puffer
390  -.eq pufstart = $f000  ;pufferanfang
400  -.eq intctrl  = $dc0e  ;vic-register zur interruptkontrolle
410  -.eq crsline  = 214    ;cursorzeile
420  -.eq linepoi  = 209    ;pointer auf cursorzeile
```

Bild 4. Verwendete Label

Zwei-Byte-Pointer auf, mit denen eine indirekt indizierte Adressierung des Bildschirms beziehungsweise des Pufferbereichs vorgenommen wird. Da mehrere Puffer verwaltet werden (Window-Überlagerung), wird eine Tabelle benötigt, die die Anfangsadressen der verschiedenen Puffer enthält. Diese Tabelle beginnt ab PUFPOI. PUFSTART kennzeichnet die Anfangsadresse des ersten Puffers ab $F000.

Wie Sie anhand der Adresse erkennen, liegt der Pufferbereich, in dem die zu rettenden Bildschirmausschnitte gespeichert werden, »unter« dem Betriebssystem. Der Grund: Ich wollte es möglichst vermeiden, kostbaren Speicherplatz (zum Beispiel im Bereich $C000 bis $CFFF) zu verschwenden. Insgesamt stehen zwei KByte an Pufferbereichen zur Verfügung.

Sprungverteiler

Bild 5 zeigt den Programmanfang. Der Programmstart wird auf die Adresse $C600 gelegt. Am Programmanfang befindet sich ein »Sprungverteiler«, der zum Aufruf der Routinen von Basic aus verwendet werden sollte. Der Vorteil: Selbst bei eventuellen Programmänderungen bleiben die SYS-Aufrufadressen für alle Routinen erhalten.

Gemeinsame Unterprogramme

Vor allem der Strukturierung wegen folgen nun (Bild 6) mehrere Unterprogramme, die von allen beschriebenen Routinen benötigt werden.

1. Parameter lesen: Diese Routine liest mit CHKKOM und GETBYT eine im Akku übergebene Anzahl von Ein-Byte-Werten (meist die Parameter spalte, zeile, breite, länge, flag) aus dem Basic-Text ein und legt sie nacheinander in COL, COL+1, COL+2,... ab.

2. Stringdescriptoren holen: Wie Sie wissen, legt der Basic-Interpreter Strings am Ende des verfügbaren Speicherbereichs ab und vermerkt in der Variablentabelle Länge und Adresse der Strings. Diese sogenannten Stringdescriptoren bestehen aus drei Byte, einem Byte für die Stringlänge und zwei Byte für den Pointer auf den String. Dem Unterprogramm »Stringdescriptoren holen« wird in INDIZ(+1) ein Pointer auf die Descriptoren eines Strings übergeben. Das Unterprogramm kopiert die drei Descriptorbyte nach STRLEN und STRPOS(+1).

3. Das folgende Unterprogramm geht ebenfalls davon aus, daß INDIZ(+1) einen Pointer auf die Descriptoren eines Strings enthält. Dieser Pointer wird um drei erhöht und weist damit auf die Descriptoren des nächsten Strings eines Stringarrays (nur im Falle von Arraystrings benutzt der Interpreter drei Byte für die Descriptoren eines Strings, bei einfachen Stringvariablen sieben Byte).

4. Das letzte dieser Unterprogramme nennt sich »Endzeile berechnen«, ein nicht gerade aussagekräftiger Name. Gemeint ist folgendes: Wie erläutert, behandeln alle Routinen einen rechteckigen Bildschirmausschnitt. Bei der Bearbeitung dieses Ausschnitts muß nach jeder Zeile geprüft werden, ob die letzte Zeile des Rechtecks bereits behandelt wurde. »Endzeile berechnen« ermittelt anhand der übergebenen Nummer (LINE) der ersten Window-Zeile und der ebenfalls übergebenen Window-Länge die letzte Window-Zeile und speichert sie in LAENGE. Zugegebenermaßen ist das Label LAENGE mißverständlich, da es nach dem Aufruf dieses Unterprogramms nicht mehr für die Window-Länge in Zeilen steht, sondern die entsprechende Speicherzelle, die die Nummer der letzten Window-Zeile enthält.

Die Mal-Routine (Window plus Inhalt malen)

Bild 7 zeigt Ihnen den Source-Code der Mal-Routine. Die Routine besteht aus vier Teilen: Einem Initialisierungsteil, der die benötigten Parameter einliest und weitere Vorbereitungen trifft, und drei Abschnitte, die die erste beziehungsweise letzte Window-Zeile und die »Innenzeilen« auf dem Bildschirm ausgeben.

Der Initialisierungsteil verwendet das Unterprogramm »Parameter lesen«, um die vier Parameter »spalte, zeile, breite, länge« aus dem Basic-Text zu lesen. Anschließend wird mit der Interpreter-Routine GETPOS ein Pointer (Y/Akku) auf die Descriptoren des übergebenen Strings ARRAY eingelesen.

Im Anschluß an diesen Teil folgt das Label WINJSR, das den Einsprung von Maschinensprache aus markiert. Voraussetzung für einen solchen Einsprung ist selbstverständlich, daß

```
460  -.ba $c600 ;programmstart
470  -;
480  -;
490  -;
500  -;*** sprungverteiler ***
510  -           jmp window
520  -           jmp puffer
530  -           jmp invert
540  -           jmp cntrl
```

Bild 5. Programmstart und Sprungverteiler

```
580  -;******************************
590  -;*** gemeinsame unterprogs ***
600  -;******************************
610  -;
620  -;*** parameter lesen ***
630  -param     sta cnti        ;liest beliebige
640  -          ldx #0          ;anzahl an
650  -lesen     stx cntj        ;ein-byte-werten
660  -          jsr chkkom      ;aus dem basic-text
670  -          jsr getbyt      ;und legt sie
680  -          txa             ;ab 'col' ab.
690  -          ldx cntj
700  -          sta col,x
710  -          inx
720  -          cpx cnti
730  -          bne lesen
740  -          rts
750  -;
760  -;
770  -;*** stringdescriptoren holen ***
780  -holdes    ldy #2          ;holt die string-
790  -holdes1   lda (indiz),y   ;descriptoren der
800  -          sta strlen,y    ;uebergebenen string-
810  -          dey             ;variablen
820  -          bpl holdes1     ;laenge/pointer auf string
830  -          rts
840  -;
850  -;
860  -;*** indiz(+1)=>next string ***
870  -nextstr   lda indiz       ;erhoeht einen pointer
880  -          clc             ;(in 'indiz') auf einen
890  -          adc #3          ;stringdescriptor, so
900  -          sta indiz       ;dass dieser auf den
910  -          bcc next1       ;naechsten descriptor
920  -          inc indiz+1     ;weist (indiz=indiz+3)
930  -next1     rts
940  -;
950  -;
960  -;*** endzeile berechnen ***
970  -endzei    lda line        ;berechnet die nummer der
980  -          clc             ;letzten zeile des angeg.
990  -          adc laenge      ;rechtecks (lastzeile=
1000 -          sta laenge      ;startzeile+laenge)
1010 -          rts
```

Bild 6. Gemeinsame Unterprogramme

das aufrufende Programm die erwähnten Parameter zuvor in den von »Parameter lesen« verwendeten Adressen ablegt und den benötigten Pointer ebenso wie die Interpreter-Routine GETPOS in Akku und Y-Register übergibt.

Dieser Pointer wird in INDIZ(+1) gespeichert, bevor die letzte Window-Zeile ermittelt wird und die Parameter »länge, breite« korrigiert werden (warum, erkennen Sie bei näherer Analyse des folgenden Hauptteils).

Das Malen einer Window-Zeile wird von einer eigenen Routine vorgenommen. Dieser Routine wird beim Malen der ersten Window-Zeile, die nur aus Grafikzeichen (erinnern Sie sich an den Rahmen, der das Window umgibt) besteht, das Flag Null übergeben. Jeder Wert ungleich eins sagt aus, daß die betreffende Zeile nur aus Grafikzeichen besteht und nicht (!) mit dem Inhalt eines Strings (einem Kommando des Untermenüs) gefüllt werden soll.

Nachdem die erste Zeile gemalt wurde, werden die »Window-Innenzeilen« ausgegeben. Die erste und letzte Spalte dieser Innenzeilen besteht aus Grafikzeichen (Rahmen), der innere Teil wird mit dem jeweiligen Kommandostring gefüllt (Flag=1). Der Aufruf der Routine »Zeile malen« erfolgt in einer Schleife solange, bis alle Innenzeilen ausgegeben wurden. Nach jedem Aufruf wird der Pointer INDIZ(+1) auf die Descriptoren des folgenden Arraystrings (des folgenden Untermenükommandos) gesetzt und die betreffende Zeile von der Routine »Zeile malen« mit diesem Kommandostring gefüllt.

Die Ausgabe der letzten Window-Zeile, die wiederum nur aus Grafikzeichen besteht, entspricht der ersten Zeile.

Den Hauptteil bildet die Routine »Zeile malen«, die für die Ausgabe einer Zeile zuständig ist. Der Ablauf: Der Cursor wird auf die aktuelle Zeile und darin auf die erste Spalte gesetzt, in der das Window beginnt. Wenn das Flag einen Wert ungleich eins besitzt, handelt es sich bei der auszugebenden Zeile um die erste oder die letzte Window-Zeile. Die Innenspalten des Windows bestehen in diesem Fall ausschließlich aus Grafikzeichen.

Soll dagegen eine Innenzeile ausgegeben und mit einem Kommandostring gefüllt werden, liest die Routine Zeichen für Zeichen dieses Strings anhand der übergebenen Stringdescriptoren und gibt ihn aus. Wird innerhalb der Ausgabeschleife das Stringende erreicht (wenn die Stringlänge kleiner ist als die Window-Länge minus zwei), wird der Rest der Window-Zeile mit Leerzeichen aufgefüllt.

Die Window-Routine (Window-Untergrund retten/holen)

Bild 8 enthält den Source-Code der Window-Routine. Nach dem Einlesen der benötigten Parameter wird der Pointer auf den ersten Puffer Nummer Null initialisiert (gleich der Puffer-Startadresse). Anschließend wird anhand der übergebenen Puffernummer der Pointer auf den gewünschten Puffer aus der Tabelle aller Puffer geholt. Im Normalfall – ohne Überlagerung mehrerer Windows – werden Sie immer den ersten Puffer (Puffer Nummer Null) verwenden und der »Pufferpointer« daher dem initialisierten Pointer auf den Anfang des Pufferbereichs entsprechen.

Nach Ermittlung der letzten Window-Zeile wird der Cursor auf die linke obere Ecke des Windows gesetzt und ein weiterer Pointer erzeugt, der auf die zugehörige Speicherzelle des Video-RAMs zeigt (=Pointer auf Zeilenanfang+Startspalte).

Nun folgt eine Schleife, die entsprechend der Window-Breite Zeichen für Zeichen entweder vom Bildschirm in den Puffer kopiert (flag=1) oder umgekehrt den Pufferinhalt in die aktuelle Bildschirmzeile schreibt.

Anschließend wird der Pointer auf den Puffer um die jeweilige Zeilenlänge (=Window-Breite) erhöht und der Cursor durch Ausgabe des Steuerzeichens Cursor down (ASCII-Code 17) auf den Beginn der nächsten Window-Zeile gesetzt. Wurde die letzte Window-Zeile in der inneren Schleife noch nicht gepuffert beziehungsweise zurückgeschrieben, wird eine weitere Zeile auf die beschriebene Art und Weise behandelt.

```
1050 -;*************************
1060 -;* window + inhalt malen *
1070 -;*************************
1080 -;
1090 -;aufruf: sys xxxx,spalte,zeile,
1100 -;        breite,laenge,array
1110 -;
1120 -;funktion: malt ein window der
1130 -;          angegebenen  groesse
1140 -;          ('breite','laenge')
1150 -;          ab der posit.'spalte'/
1160 -;          'zeile' und fuellt die
1170 -;          innenzeilen mit den
1180 -;          angegeb. arraystrings,
1182 -;          deren anzahl mindes-
1184 -;          tens ebenso gross wie
1186 -;          die anzahl der innen-
1188 -;          zeilen sein muss.
1190 -;
1200 -;bsp.: sys xxxx,2,5,10,15,a$(2)
1210 -;      malt window mit linker
1220 -;      ecke 2/5, das 10 spalten
1230 -;      breit und 15 zeilen lang
1240 -;      ist; die innenzeilen wer-
1250 -;      den mit 'a$(2)'-'a$(15)'
1260 -;      gefuellt
1270 -;
1280 -;
1290 -window    lda #4          ;4 bytewerte holen
1300 -          jsr param
1310 -;
1320 -          jsr chkkom      ;stringdescrriptor
1330 -          jsr getpos      ;holen
1340 -;
1350 -winjsr    sta indiz       ;und nach 'indiz(+1)'
1360 -          sty indiz+1     ;schaffen
1370 -;
1380 -          jsr endzei      ;letzte windowzeile
1390 -          dec laenge      ;korrektur
1400 -          dec breite      ;korrek-
1410 -          dec breite      ;tur
1420 -;
1430 -;
1440 -;*** erste zeile malen ***
1450 -          lda #0          ;flag=0 =>
1460 -          sta flag        ;ohne string ausgeben
1470 -          jsr malen
1480 -;
1490 -;
1500 -;*** innenzeilen malen ***
1510 -          inc flag        ;flag=0 =>
1520 -innen     jsr holdes      ;zeilen mit inhalten
1530 -          jsr malen       ;der arraystrings fuellen
1540 -          jsr nextstr
1550 -          lda line        ;alle innenzeilen
1560 -          cmp laenge      ;ausgegeben?
1570 -          bne innen       ;nein =>
1580 -;
1590 -;
1600 -;*** letzte zeile malen ***
1610 -          inc flag        ;flag=2 =>
1620 -          jmp malen       ;zeile ohne stringinhalte
1630 -;
1640 -;
1650 -;*** zeile malen ***
1660 -malen     ldx line        ;cursor setzen
1670 -          ldy col
1680 -          clc
1690 -          jsr plot
1700 -          ldx flag
1710 -          lda links,x     ;linkes zeichen
1720 -          jsr bsout       ;ausgeben
1730 -          ldy #0          ;mittleres zeichen
1740 -loop      lda mitte,x     ;laden und ausgeben,
1750 -          cpx #1          ;wenn flag
1760 -          bne nostring    ;<>1 =>
1770 -          cpy strlen      ;oder stringende
1780 -          bcs nostring    ;erreicht =>
1790 -          lda (strpos),y  ;sonst stringzeichen
1800 -nostring  jsr bsout       ;ausgeben
1810 -          iny             ;zeile -1 zeichen
1820 -          cpy breite      ;komplett ausgegeben?
1830 -          bne loop        ;nein =>
1840 -          inc line        ;next zeile
1850 -          lda rechts,x    ;rechtes zeichen
1860 -          jmp bsout       ;ausgeben + rts !!!
1870 -;
1880 -links     .by 176,125,173
1890 -mitte     .by 96,32,96
1900 -rechts    .by 174,125,189
```

Bild 7. Window und Inhalt auf den Bildschirm bringen

Nachdem das komplette Window behandelt wurde, weist der Pointer PUFFERP(+1) exakt auf das Ende des aktuellen Puffers. Dadurch ist automatisch die Startadresse des nächsten Puffers festgelegt. Angenommen, $F100 markiert das Ende von Puffer Nummer Null. Puffer Nummer Eins beginnt in diesem Fall genau ein Byte weiter, ab $F101. Diese Startadresse wird zum Abschluß der Routine in die Tabelle der Puffer-Startadressen eingetragen. Sie verstehen nun sicherlich, was ich meine, wenn ich von »dynamischer« Pufferverwaltung spreche.

Die Invertier-Routine (Ausschnitt invertieren/normalisieren)

Die Invertier-Routine (Bild 9) ist geradezu »primitiv«, verglichen mit der Window-Routine. Wie üblich, werden zuerst Parameter gelesen und die letzte Window-Zeile berechnet.

Zu Beginn der äußeren von zwei ineinandergeschachtelten Schleifen wird der Cursor auf die obere linke Window-Ecke gesetzt. Durch die Addition der Cursor-Spalte und des vom





Betriebssystem verwalteten Pointers auf den Beginn der aktuellen Cursor-Zeile, wird ein Pointer auf die momentane Cursor-Position erzeugt.

In der folgenden inneren Schleife »hangelt« sich die Routine entlang der einzelnen Zeichen, aus denen die aktuelle Window-Zeile besteht. Jedes dieser Zeichen wird invertiert (Bit sieben setzen) oder normalisiert (Bit sieben löschen), je nach übergebenem Flagzustand (flag=1 => invertieren; flag=0 => normalisieren).

Nachdem die innere Schleife durchlaufen wurde, wird der »Zeilenzähler« LINE inkrementiert und enthält somit die Nummer der folgenden Bildschirmzeile, bevor wiederum zum Beginn der äußeren Schleife verzweigt wird.

Die Routine wird verlassen, wenn die Bedingung LINE=LAENGE erfüllt ist. In diesem Fall wurde die letzte Window-Zeile behandelt.

Hauptprogramm (Verwaltung von Pull-down-Menüs)

Nach diesem »Vorgeplänkel« nähern wir uns dem Ziel, der Verwaltung von Pull-down-Menüs. Das notwendige »Handwerkszeug« ist vorhanden, es fehlt jedoch noch ein Steuerungsprogramm, das für eine »intelligente« Nutzung der Routinen sorgt.

Das Hauptprogramm ist ziemlich umfangreich, so daß ich es abschnittsweise besprechen werde.

Bild 10 zeigt fünf zusätzliche Label, die in der Routine verwendet werden:

1. MCOUNT stellt einen Zähler dar, der die Nummer des jeweils aktiven Untermenüs enthalten wird (ab null).
2. MZAHL enthält die Gesamtanzahl der zu verwaltenden Untermenüs.
3. POINT ist wiederum ein Zähler, der den jeweils selektierten Menüpunkt (=Kommando) angibt (ab null).
4. ZEICHEN wird zur Tasten-Speicherung verwendet und enthält den ASCII-Code der vom Benutzer betätigten Taste.
5. VEKTOR: Die verschiedenen Unterroutinen zur Verarbeitung von Benutzereingaben (<CRSR>-rechts/-links) werden über einen indirekten Sprung aufgerufen (JMP (VEKTOR)).

Bild 11 enthält den ersten abgeschlossenen Teil dieses Hauptprogramms, die Ermittlung aller benötigten Parameter des gesamten Pull-down-Menüs. Folgende Parameter benötigt das Steuerungsprogramm:

- Die Anfangsbuchstaben der verschiedenen Untermenü-Namen.
- Die Window-Parameter »spalte,breite,länge«.
- Für jedes Untermenü einen Pointer auf den ersten Kommandostring dieses Menüs.
- Da der Name des aktiven Untermenüs invertiert werden soll, ist es sinnvoll, zu Beginn die Breite dieser Namen zu ermitteln. Diese Angabe wird beim Aufruf der Invertier-Routine benötigt, um die Breite des zu invertierenden Rechtecks anzugeben.
- Die Anzahl der zu verwaltenden Untermenüs.

Sollten Sie einen wichtigen Parameter vermissen, die Nummer der jeweils ersten Menüzeile: Diese Angabe steht bereits fest, da wir vereinbarten, daß sich die Menüleiste in Zeile Nummer Null befindet und die Untermenüs unterhalb der Menüleiste beginnen. Daher ist die Zeile eins die erste Zeile aller Untermenüs.

Diese Parameter (mit Ausnahme der Untermenü-Anzahl) werden in Tabellen am Programmende abgelegt (Bild 12). Die in den Tabellen verwendeten Label besitzen folgende Bedeutungen:

- START: Tabelle der Menü-Startspalten.
- LENGTH: Tabelle der Länge der verschiedenen Menünamen.
- SIGN: Anfangsbuchstaben der Menünamen.
- WBREITE: Breite der Untermenüs.
- WLAENGE: Länge der Untermenüs.

```
1940 -;*********************************
1950 -;* windowuntergrund retten/holen*
1960 -;*********************************
1970 -;
1980 -;aufruf:sys xxxx,spalte,zeile,
1990 -;        breite,laenge,flag,puffer
2000 -;
2010 -;funktion: kopiert den durch
2020 -;           'spalte'/'zeile' und
2030 -;           'breite' bzw.'laenge'
2040 -;           angegebenen bild-
2050 -;           schirmausschnitt in
2060 -;           puffer nr.'puffer',
2070 -;           wenn 'flag'=1.
2080 -;           'flag'=1 =>der inhalt
2090 -;           des angeg.puffers wird
2100 -;           in den angeg. bild-
2110 -;           schirmausschnitt
2120 -;           kopiert.
2122 -;           puffer: numerierung
2124 -;           beginnt bei 0; max.
2126 -;           pufferbereich: 2 kb
2128 -;
2130 -;bsp.: sys xxxx,2,5,10,15,1,0
2140 -;       window mit linker oberer
2150 -;       ecke 2/5, breite 10 spal-
2160 -;       ten u.laenge 15 zeilen in
2170 -;       puffer nr.0 schreiben.
2180 -;
2190 -;
2200 -puffer     lda #6          ;6 bytewerte lesen
2210 -           jsr param
2220 -;
2230 -pufferjsr  lda <(pufstart);pointer auf
2240 -           ldx >(pufstart);puffer nr.0
2250 -           sta pufpoi      ;erzeugen
2260 -           stx pufpoi+1
2270 -;
2280 -           ldx pufnr       ;adresse des angeg.
2290 -           lda pufpoi,x    ;puffers nach
2300 -           sta pufferp     ;'pufpoi(+1)'
2310 -           lda pufpoi+1,x  ;kopieren
2320 -           sta pufferp+1
2330 -;
2340 -           jsr endzei      ;letzte zeile ermitteln
2350 -;
2360 -           ldx line        ;cursor auf line
2370 -           ldy col
2380 -           clc             ;obere windowecke
2390 -           jsr plot
2400 -;
2410 -weiter     jsr noint       ;interrupt ausschalten
2420 -           lda #$34        ;ram-konfoguration
2430 -           sta 1           ;einstellen
2440 -           jsr intein      ;interrupt einschalten
2450 -           lda linepoi+1   ;'screenpointer(+1)'
2460 -           sta screenp+1   ;+windowstartspalte
2470 -           lda linepoi     ;ergibt pointer
2480 -           clc             ;auf anfang der
2490 -           adc col         ;aktuellen windowzeile
2500 -           sta screenp     ;+startspalte
2510 -           bcc okay
2520 -           inc screenp+1
2530 -;
2540 -okay       ldy breite      ;zaehler initialiseren
2550 -           dey
2560 -copy       lda flag        ;ein zeichen aus puffer
2570 -           bne write       ;auf screen oder
2580 -           lda (screenp),y;umgekehrt,
2590 -           sta (pufferp),y;je nach flagzustand
2600 -           bne jump
2610 -write      lda (pufferp),y
2620 -           sta (screenp),y
2630 -jump       dey             ;zeile behandelt?
2640 -           bpl copy        ;nein =>
2650 -;
2660 -           lda pufferp     ;pointer auf
2670 -           clc             ;puffer um
2680 -           adc breite      ;windowbreite
2690 -           sta pufferp     ;erhoehen
2700 -           bcc noincr
2710 -           inc pufferp+1
2720 -;
2730 -noincr     jsr noint       ;interrupt aus
2740 -           lda #$37        ;rom-konfiguration
2750 -           sta 1           ;einschalten
2760 -           jsr intein      ;interrupt ein
2770 -           lda #17         ;cursor eine zeile
2780 -           jsr bsout       ;tiefer setzen
2790 -           lda crsline     ;letzte windowzeile
2800 -           cmp laenge      ;behandelt?
2810 -           bne weiter      ;nein =>
2820 -;
2830 -;
2840 -           ldx pufnr       ;anfang des
2850 -           inx             ;naechsten puffers
2860 -           txa             ;hinter das ende
2870 -           asl             ;des aktuellen
2880 -           tax             ;puffers setzen
2890 -           lda pufferp
2900 -           sta pufpoi,x
2910 -           lda pufferp+1
2920 -           sta pufpoi+1,x
2930 -           rts
2940 -;
2950 -;
2960 -noint      lda intctrl     ;interrupts verhindern
2970 -           and #$fe
2980 -           sta intctrl
2990 -           rts
3000 -;
3010 -intein     lda intctrl     ;interrupts zulassen
3020 -           ora #$01
3030 -           sta intctrl
3040 -           rts
```

Bild 8. Window-Hintergrund retten und holen

- WPOILOW: Low-Byte des Pointers auf den jeweils ersten Kommandostring der Untermenüs.
- WPOIHIGH: High-Byte des Pointers auf den jeweils ersten Kommandostring der Untermenüs.

Um die Parameter-Ermittlung zu verstehen, müssen Sie wissen, in welcher Form die Kommandostrings und die Menüleiste an das Steuerungsprogramm übergeben wer-

```
3080 -;********************************
3090 -;* screenausschnitt invertieren *
3100 -;********************************
3110 -;
3120 -;aufruf: sys xxxx,spalte,zeile,
3130 -;          breite,laenge,flag
3140 -;
3150 -;funktion: invertiert('flag'=1)/
3160 -;           normalisiert('flag'=0)
3170 -;           einen rechteckigen
3180 -;           bildschirmausschnitt
3190 -;           mit der oberen linken
3200 -;           ecke 'spalte'/'zeile'
3210 -;           und der angegebenen
3220 -;           laenge bzw. breite.
3230 -;
3240 -;bsp.: sys xxxx,2,5,10,15,1
3250 -;       invertiert ein rechteck
3260 -;       mit der linken obereb ecke
3270 -;       2/7, der breite 10 spalten
3280 -;       und der laenge 15 zeilen.
3290 -;
3300 -;
3310 -invert    lda #5          ;5 bytewerte holen
3320 -          jsr param
3330 -;
3340 -invjsr    jsr endzei      ;letzte rechteckzeile
3350 -          dec breite      ;korrektur
3360 -          ldx line        ;cursor auf aktuelle
3370 -;
3380 -inv2      clc             ;zeile und startspalte
3390 -          jsr plot        ;setzen
3400 -;
3410 -          lda linepoi     ;'linepoi' + 'col' =>
3420 -          clc             ;pointer auf
3430 -          adc col         ;erstes zu behan-
3440 -          sta linepoi     ;delndes zeichen
3450 -          bcc inv3        ;zeichen der jewei-
3460 -          inc linepoi+1   ;ligen zeile
3470 -;
3480 -inv3      ldy breite      ;wenn flag=0, wird
3490 -inv1      lda (linepoi),y;die komplette
3500 -          ldx flag        ;zeile normalisiert
3510 -          beq norm        ;(bit 7 loeschen),
3520 -          ora #$80        ;sonst invertiert
3530 -          .by $2c         ;(bit 7 seetzen)
3540 -norm      and #$7f
3550 -          sta (linepoi),y
3560 -          dey             ;zeile behandelt?
3570 -          bpl inv1        ;nein =>
3580 -;
3590 -          inc line        ;next zeile
3600 -          ldx line
3610 -          cpx laenge      ;letzte zeile behandelt?
3620 -          bne inv2        ;nein =>
3630 -          rts
```

Bild 9. Bildschirmausschnitt invertieren

den. Betrachten Sie bitte das in Basic geschriebene »Demoprogramm 1« (Listing 1) am Ende dieses Kapitels.

Der Aufruf der Routine lautet prinzipiell SYS xxxx,leiste$,array$. Im Demoprogramm wird der Leistenstring <K$> verwendet, der die Untermenü-Namen FILE, DISK, EDIT und HELP enthält.

Das Array <A$(...)> enthält alle Kommandos der vier Untermenüs. Vor jedem Untermenü befindet sich ein String, der ausschließlich aus Leerzeichen (Spaces) besteht und dessen Länge der gewünschten Menübreite (ohne Rahmen) entspricht, gewöhnlich der Zeichenanzahl des längsten Kommandos im betreffenden Untermenü.

Nach diesem Schema (Leerstring, Kommandostrings) sind alle vier Untermenüs aufgebaut. Dem letzten Kommandostring muß wiederum ein Leerstring folgen, dessen Länge jedoch beliebig ist (siehe Zeile 410 in »Demoprogramm 1«).

Die Parameter werden nun wie folgt ermittelt:

1. Die Descriptoren des Menüleistenstrings werden gelesen und zur späteren Verwendung auf den Stack kopiert (»Menüparameter holen«).

2. Die einzelnen Zeichen des Leistenstrings werden gelesen (»Startspalten der Menüs«), das heißt die Menünamen, zwischen denen sich zur Unterscheidung mindestens ein Leerzeichen befinden sollte. Während dieses Vorgangs werden bereits drei Parameter ermittelt und in den Tabellen abgelegt: die Anfangsbuchstaben der Menünamen (Tabelle SIGN), die Startspalten der Menünamen und damit zugleich die Startspalten der zugehörigen Untermenüs (Tabelle START) und die Länge der Menünamen (Tabelle LENGTH). Wenn dieser Vorgang beendet ist, das heißt wenn das letzte Zeichen des Leistenstrings gelesen wurde, steht zugleich die Anzahl MZAHL der Untermenüs fest.

3. Der Programmteil »Window-Parameter holen« ermittelt die noch fehlenden Window-Parameter WLAENGE und WBREITE und den Pointer WPOILOW/WPOIHIGH, der auf die Adresse des jeweils ersten Untermenüstrings zeigt. WBREITE entspricht der Länge des erwähnten Leerstrings,

```
3670 -;********************************
3680 -;* verwaltung v.pull-down-menues*
3690 -;********************************
3700 -;
3710 -;aufruf:sys xxxx,menue$,menue$(1)
3720 -;
3730 -;funktion: verwaltet beliebig
3740 -;           viele pull-d.menues,
3750 -;           deren oberbegriffe in
3760 -;           'menue$'enthalten sind
3770 -;           (z.b.:a$='file edit').
3780 -;           die menuekommandos
3790 -;           enthaelt das angegeb.
3800 -;           stringarray. vor jedem
3810 -;           untermenue befindet
3820 -;           sich ein leerstring
3830 -;           mit der gewuenschten
3840 -;           menuelaenge.
3850 -;           dem letzten menue-
3860 -;           string folgt ebenfalls
3870 -;           ein leerstring.
3880 -;
3890 -;bsp.: sys xxxx,a$,a$(1)
3900 -;       verwaltet z.b. folgende
3910 -;       verwaltet 2 untermenues
3920 -;       mit 4 bzw. 7 auswahlpkten,
3930 -;       wenn folgendes gilt:
3935 -;
3940 -;        a$='document disk'
3950 -;        a$(1)='      '
3960 -;        a$(2)='load'
3970 -;        a$(3)='save'
3980 -;        a$(4)='merge'
3990 -;        a$(5)='print'
4000 -;        a$(6)='
4010 -;        a$(7)='directory'
4012 -;        a$(8)='initialise'
4014 -;        a$(9)='validate'
4020 -;       a$(10)='format disk'
4030 -;       a$(11)='scratch file'
4040 -;       a$(12)='copy file'
4050 -;       a$(13)='rename file'
4060 -;       a$(14)=' '
4070 -;
4080 -;bedienung: cursor right/left =>
4090 -;            auswahl untermenue
4100 -;            cursor down/up =>
4110 -;            auswahl im untermenue
4120 -;            return =>
4130 -;            auswahl aktueller
4150 -;            menuepunkt
4160 -;
4170 -;            ein untermenue kann
4180 -;            alternativ mit shift+
4190 -;            dem anfangsbuchstaben
4200 -;            des menuenamens
4210 -;            direkt angewaehlt
4220 -;            werden, z.b. shift+f
4230 -;            fuer d.menue 'file'.
4240 -;            ein menuepunkt kann
4250 -;            alternativ mit dem
4260 -;            anfangsbuchstaben
4270 -;            direkt gewaehlt wer-
4280 -;            den('s'fuer'scratch')
4281 -;
4282 -;            das angewaehlte menue
4283 -;            wird in 'mcount', der
4284 -;            gewaehlte menuepunkt
4285 -;            in 'point' uebergeben
4290 -;
4300 -;bsp.: sys xxxx,a$,a$(1) ver-
4310 -;       waltet das oben verwendete
4320 -;       beispiel
4330 -;
4340 -;
4350 -.eq mcount  = routin+1  ;zaehler: aktuelles menue
4360 -.eq mzahl   = mcount+1  ;gesamte menueanzahl
4370 -.eq point   = mzahl+1   ;zaehler: aktueller menuepunkt
4380 -.eq zeichen = point+1   ;gedrueckte taste
4390 -.eq vektor  = zeichen+1 ;vektor f.indir.jmp
```

Bild 10. Verwaltung der Pull-down-Menüs

WLAENGE wird ermittelt, indem sich die Routine String für String bis zum nächsten Leerstring vortastet, der ja den Beginn eines weiteren Untermenüs kennzeichnet. Der Pointer WPOILOW/WPOIHIGH zeigt auf die Descriptoren jenes Strings, der dem Leerstring folgt, das heißt auf den jeweils ersten Kommandostring der Untermenüs.

Bild 13 zeigt den nächsten Programmteil, die Initialisierung. Der Cursor wird auf die HOME-Position gesetzt und die Menüleiste ausgegeben. Zuletzt wird der Menüzähler MCOUNT initialisiert.

Der Teil »Menü-Ausgabe« initialisiert nun das erste (MCOUNT=0) Pull-down-Menü. Der Untergrund des zu zeichnenden Menüs wird mit der Window-Routine gerettet (JSR LIESGR), der Menüname – im Basic-Demoprogramm der Name FILE – wird invertiert (JSR MINVERT), das Menü ausgegeben (JSR MPRINT) und der Kommandozähler POINT initialisiert, wobei der Wert 255 bedeutet, daß momentan kein Kommando selektiert ist.

Die folgende Tastaturabfrage (Bild 14) vergleicht ein eingegebenes Zeichen mit der Tabelle KEY, die die ASCII-Codes der vier Cursor-Tasten und der <RETURN>-Taste enthält (Bild 11). Wurde eine dieser Tasten betätigt, wird mit Hilfe der Tabelle TAB ein Vektor auf die entsprechende Routine erzeugt, und es erfolgt ein indirekter Sprung über diesen Vektor (JMP (VEKTOR)).

```
4420 -;*** menueparameter holen ***
4430 -cntrl     jsr chkkom      ;pointer auf descriptoren
4440 -          jsr getpos      ;des menueleistenstrings
4450 -          sta indiz       ;nach 'indiz(+1)' und
4460 -          sty indiz+1     ;descriptoren selbst nach
4470 -          jsr holdes      ;'strlen',strpos(+1)'
4480 -;
4490 -          lda strlen      ;descriptoren retten
4500 -          pha
4510 -          lda strpos
4520 -          pha
4530 -          lda strpos+1
4540 -          pha
4550 -;
4560 -;
4570 -;*** startspalten d.menues ***
4580 -          ldy #0          ;spalte und anzahl
4590 -          ldx #0          ;initialisieren
4600 -start1    lda (strpos),y ;im leistenstring alle
4610 -          cmp #" "        ;spaces bis zum
4620 -          bne start2      ;1.nicht-space
4630 -          iny             ;ueberlesen
4640 -          bne start1      ;immer !!!
4650 -;
4660 -start2    sta sign,x      ;anfangsbuchstabe des menues
4670 -          tya             ;y=position von 1.nichtspace
4680 -          sta start,x     ;=startcol fuer invert
4690 -start3    lda (strpos),y ;nun alle
4700 -          cmp #" "        ;nichtspaces
4710 -          beq start4      ;ueberlesen
4720 -          iny             ;jedoch nur, wenn
4730 -          cpy strlen      ;stringlaenge noch
4740 -          bcc start3      ;nicht ueberschritten
4750 -;
4760 -start4    tya             ;invertierbreite steht nun
4770 -          sec             ;fest: invertbreite=
4780 -          sbc start,x     ;y-startcol
4790 -          sta length,x    ;ergebnis in tabelle speichern
4800 -          inx
4810 -          iny             ;weiter, wenn ende des
4820 -          cpy strlen      ;leistenstrings noch nicht
4830 -          bcc start1      ;erreicht =>
4840 -;
4850 -          stx mzahl       ;menueanzahl speichern
4860 -;
4870 -;
4880 -;*** windowparameter holen ***
4890 -          jsr chkkom      ;pointer auf descriptoren
4900 -          jsr getpos      ;des 1.menuestrings holen
4910 -          sta indiz       ;und descritoren selbst
4920 -          sty indiz+1     ;nach 'strlen','strpos(+1)'
4930 -          jsr holdes      ;schaffen
4940 -;
4950 -          ldx #255        ;x initialisieren
4960 -          bne par5        ;immer !!!
4970 -;
4980 -par1      lda #0          ;stringzaehler
4990 -          sta cnti        ;initialisieren
5000 -;
5010 -par2      ldy #0          ;pointer auf stringzeichen init.
5020 -par3      lda (strpos),y ;aktuelles
5030 -          cmp #" "        ;stringzeichen=space?
5040 -          beq par4        ;ja =>
5050 -;
5060 -          jsr nextstr     ;naechsten string
5070 -          jsr holdes      ;descriptoren holen
5080 -          inc cnti        ;stringzaehler inkrem.
5090 -          bne par2        ;immer !!!
5100 -;
5110 -par4      iny             ;zeichenpointer inkrem.
5120 -          cpy strlen      ;stringende erreicht?
5130 -          bcc par3        ;nein =>
5140 -;
5150 -          lda cnti        ;laenge des aktuellen
5160 -          sta wlaenge,x   ;windows in tabelle
5170 -;
5180 -par5      inx             ;breite des aktuellen
5190 -          lda strlen      ;windows ebenfalls
5200 -          sta wbreite,x   ;in tabelle merken
5210 -;
5220 -          jsr nextstr     ;next string
5230 -          jsr holdes      ;descriptoren holen
5240 -          lda indiz       ;adresse der descriptoren
5250 -          sta wpoilow,x   ;in tabelle merken
5260 -          lda indiz+1
5270 -          sta wpoihigh,x
5280 -;
5290 -          cpx mzahl       ;alle menues durch?
5300 -          bcc par1        ;nein =>
```

Bild 11. In diesem Programmteil werden alle erforderlichen Parameter ermittelt

```
7550 -;*** kommando-tasten ***
7560 -key       .by 29          ;crs.right
7580 -          .by 157
7590 -          .by 17          ;crs.down
7600 -          .by 145         ;crs.up
7610 -          .by 13          ;return
7620 -;
7630 -;
7640 -;*** sprungtabelle ***
7660 -tab       .wo right
7670 -          .wo left
7680 -          .wo down
7690 -          .wo up
7700 -          .wo return
7710 -;
7720 -;
7725 -;*** windowparam.-tabellen ***
7730 -start     .by 1,2,3,4,5,6,7,8,9
7740 -length    .by 1,2,3,4,5,6,7,8,9
7750 -sign      .by 1,2,3,4,5,6,7,8,9
7760 -wbreite   .by 1,2,3,4,5,6,7,8,9
7770 -wlaenge   .by 1,2,3,4,5,6,7,8,9
7780 -wpoilow   .by 1,2,3,4,5,6,7,8,9
7790 -wpoihigh  .by 1,2,3,4,5,6,7,8,9
8000 -.en
```

Bild 12. Hier werden die Parameter der Menüleiste gespeichert

Wurde keine dieser Tasten betätigt, prüft der Teil »Direktanwahl« (Bild 15), ob ein Untermenü direkt angewählt wird, das heißt ob der Anfangsbuchstabe eines Menünamens eingegeben und zugleich die Taste <CTRL> gedrückt wurde.

```
5330 -;*** initialisierung ***
5340 -          lda #19         ;'cursor home
5350 -          jsr bsout       ;ausgeben
5360 -;
5370 -          pla             ;descriptoren des
5380 -          sta strpos+1    ;leistenstrings
5390 -          pla             ;holen
5400 -          sta strpos
5410 -          pla
5420 -          sta strlen
5430 -;
5440 -          ldy #0          ;leistenstring ausgeben
5450 -init1     lda (strpos),y
5460 -          jsr bsout
5470 -          iny
5480 -          cpy strlen
5490 -          bcc init1
5500 -;
5510 -          ldx #0          ;menuezaehler
5520 -          stx mcount      ;initialisieren
5530 -;
5540 -;
5550 -;*** menue-ausgabe ***
5560 -ausgabe   jsr liesgr      ;untergrund retten
5570 -          jsr minvert     ;menuename invertieren
5580 -          jsr mprint      ;menue ausgeben
5590 -          lda #255        ;menuepunkt initialisieren
5600 -          sta point       ;(255=kein punkt angewaehlt)
```

Bild 13. Das Programm wird initialisiert

Wenn ja, wird der ursprüngliche Menüuntergrund aus dem Puffer geholt, der Menüname wieder normalisiert und anschließend zum beschriebenen Programmteil »Menüausgabe« gesprungen, der das angewählte Untermenü ausgibt.

Wenn nein, wird die gedrückte Taste mit den Anfangsbuchstaben der Kommandos des aktiven Untermenüs verglichen. Eine Übereinstimmung bedeutet, daß der Benutzer das betreffende Kommando direkt anwählt. In diesem Fall werden die Menü- und die Kommandonummer in den Speicherzellen 167 und 168 an das aufrufende Programm übergeben und das Steuerungsprogramm beendet.

Der folgende Abschnitt behandelt die Cursor- und die <RETURN>-Taste. <RETURN> bewirkt ebenfalls das Verlassen des Programms, nachdem zuvor die getroffene Auswahl an das aufrufende Programm übergeben wird.

<CRSR>-rechts/-links schließen das aktuelle und aktivieren ein benachbartes Untermenü. Der gerettete Untergrund des aktuellen Menüs wird auf den Bildschirm zurückgeschrieben (JSR HOLGR), der Menüname wieder normalisiert (JSR MNORMAL) und der Menüzähler MCOUNT in- (<CRSR>-rechts) beziehungsweise dekrementiert (<CRSR>-links). Den Abschluß bildet der Sprung zur Routine AUSGABE, die das nun aktivierte Untermenü behandelt.

Die Programmteile zur Behandlung von <CRSR>-unten/-oben normalisieren das zuletzt selektierte Menükommando und invertieren das neu selektierte Kommando, bevor die Rückkehr zur Eingabeschleife erfolgt.

Bild 16 zeigt den letzten Programmabschnitt, der aus den Unterprogrammen besteht, die das beschriebene Hauptprogramm aufruft. Gemeinsam ist diesen Routinen, daß ihre Hauptarbeit vorwiegend in der Parameterübergabe an die drei Unterprogramme »Untermenü zeichnen«, »Window retten/holen« und »Screen-Ausschnitt invertieren/normalisieren« besteht. Der Aufruf dieser Unterprogramme erfolgt über die erläuterten Einsprungpunkte für Maschinenprogramme (INVJSR, WINJSR und PUFFERJSR). Zur Ermittlung der Window-Parameter werden die erwähnten Tabellen verwendet.

1. »Menüname invertieren/normalisieren« invertiert/normalisiert in der Menüleiste den Namen des Menüs Nummer MCOUNT.

2. »Menüpunkt invertieren/normalisieren« invertiert/normalisiert das durch POINT angegebene Kommando des Menüs MCOUNT.

3. »Menü ausgeben« gibt das Menü Nummer MCOUNT aus.

4. »Untergrund retten/holen« holt/rettet den Untergrund von Menü MCOUNT.

5. »Window-Param. übergeben« übernimmt für einige die-

```
5630 -;*** tastatur abfragen ***
5640 -get         jsr getin       ;auf taste
5650 -            beq get         ;warten
5660 -            sta zeichen     ;zeichen merken
5670 -;
5680 -            ldx #4          ;cursor- oder return-taste
5690 -get1        cmp key,x       ;gedrueckt?
5700 -            beq get2        ;ja =>
5710 -            dex             ;ansonsten zum
5720 -            bpl get1        ;teil 'direktanwahl'
5730 -            bmi direkt
5740 -;
5750 -get2        txa             ;zaehler mit 2
5760 -            asl             ;multiplizieren
5770 -            tax             ;ergibt zeiger auf
5780 -            lda tab,x       ;sprungtabelle
5790 -            sta vektor      ;pointer fuer
5800 -            lda tab+1,x     ;inirekten sprung
5810 -            sta vektor+1    ;erzeugen
5820 -            jmp (vektor)    ;jmp indirekt
```

Bild 14. Routine zur Tastaturabfrage

```
5850 -;*** direktanwahl? ***
5860 -direkt     cmp #$40        ;shift-tastenkombination?
5870 -           bcs direkt2     ;ja =>
5880 -           ora #$40        ;in grossbuchstabe wandeln
5890 -           ldx mzahl       ;und mit anfangsbuchstaben
5900 -           dex             ;der menuenamen
5910 -direkt1    cmp sign,x      ;vergleichen
5920 -           beq menue       ;gleich =>
5930 -           dex
5940 -           bpl direkt1
5950 -           bmi get         ;falscher buchstabe =>
5960 -;
5970 -menue      txa             ;x (zeiger auf
5980 -           pha             ;menue-comand) retten
5990 -           jsr holgr       ;alten untergrund holen
6000 -           jsr mnormal     ;alten menuenamen normalisieren
6010 -           pla             ;x wiederholen und
6020 -           tax             ;nach 'crs.left/right' springen
6030 -           bpl mokay       ;immer !!!
6040 -;
6050 -direkt2    ldx mcount      ;pointer auf 1.string
6060 -           lda wpoilow,x   ;des aktuellen
6070 -           sta indiz       ;menues nach
6080 -           lda wpoihigh,x  ;'indiz(+1)'
6090 -           sta indiz+1     ;holen
6100 -;
6110 -           lda #0          ;zaehler intialisieren
6120 -direkt3    tax
6130 -           jsr holdes      ;descriptoren holen
6140 -           iny             ;taste mit den anfangs-
6150 -           lda (strpos),y  ;buchstaben der menue-
6160 -           cmp zeichen     ;strings vergleichen
6170 -           beq direkt4     ;gleich =>
6180 -           jsr nextstr     ;sonst next string
6190 -           inx             ;untersuchen, wenn
6200 -           txa             ;letzter menue-
6210 -           ldx mcount      ;string noch nicht
6220 -           cmp wlaenge,x   ;behandelt wurde
6230 -           bcc direkt3
6240 -           bcs get         ;buchstabe nicht im menue vorhanden =>
6250 -;
6260 -direkt4    stx point       ;menuepunkt retten
6270 -return     lda mcount      ;menuenummer und
6280 -           sta parback     ;menuepunkt an basic
6290 -           lda point       ;uebergeben
6300 -           sta parback+1
6310 -           jsr mnormal     ;menuename normalisieren
6315 -           jmp holgr       ;untergrund holen + rts !!! => basic !!!!!!
6320 -;
6330 -right      jsr holgr       ;untergrund holen
6340 -           jsr mnormal     ;menuename normalisieren
6350 -           ldx mcount      ;wenn bereits letztes
6360 -           inx             ;menue erreicht: zaehler
6370 -           cpx mzahl       ;auf 1.menue, sonst
6380 -           bcc mokay       ;zaehler inkrem.
6390 -           ldx #0
6400 -           beq mokay       ;immer !!!
6410 -;
6420 -left       jsr holgr       ;untergrund holen
6430 -           jsr mnormal     ;menuename normalisieren
6440 -           ldx mcount      ;wenn zaehler bereits
6450 -           dex             ;auf 1.menue: zaehler
6460 -           cpx #255        ;auf last menue,
6470 -           bne mokay       ;sonst zaehler
6480 -           ldx mzahl       ;dekrem.
6490 -           dex
6500 -;
6510 -mokay      stx mcount      ;menuenummer retten
6520 -           jmp ausgabe     ;menue ausgeben + rts !!!
6530 -;
6540 -down       lda point       ;naechstes menue, wenn
6550 -           bmi down1       ;letztes menue nicht
6560 -           jsr comnormal   ;bereits erreicht,
6570 -down1      inc point       ;sonst 1.menue
6580 -           ldx mcount
6590 -           lda point
6600 -           cmp wlaenge,x
6610 -           bcc down2
6620 -           lda #0
6630 -           sta point
6640 -down2      jsr cominvert   ;aktuellen menuepunkt invertieren
6650 -down3      jmp get         ;=> eingabeschleife
6660 -;
6670 -up         lda point       ;voriger menuepunkt,
6680 -           bmi down3       ;wenn 1.menuepunkt nicht
6690 -           jsr comnormal   ;bereits erreicht,
6700 -           lda point       ;sonst last menuepunkt
6710 -           bne up1
6720 -           ldx mcount
6730 -           lda wlaenge,x
6740 -           sta point
6750 -up1        dec point
6760 -           jmp down2       ;weiter wie bei 'down'
```

Bild 15. Wurde keine Taste gedrückt, überprüft dieser Programmteil, ob ein Untermenü direkt angewählt wurde

ser Unterroutinen die »Schwerarbeit«, die Parameter START, WBREITE und WLAENGE aus den Tabellen zu ermitteln. Ausgangspunkt ist auch in dieser Routine wiederum die Menünummer MCOUNT.

```
6785 -;*** menuename invert./normal.***
6790 -comnormal lda #0          ;normalisieren:
6800 -          .by $2c         ;flag=0
6820 -cominvert lda #1          ;invertieren:
6830 -          sta flag        ;flag=1
6840 -com1      lda #2
6850 -          clc             ;invertierparameter
6860 -          adc point       ;(line,col,breite,
6870 -          sta line        ;laenge) fuer den
6880 -          ldx mcount      ;aktuellen menue-
6890 -          lda start,x     ;punkt setzen und
6900 -          sta col         ;die invertier-
6910 -          inc col         ;routine aufrufen
6920 -          lda wbreite,x
6930 -          sta breite
6940 -          lda #1
6950 -          sta laenge
6960 -          jmp invjsr      ;rts !!!
6970 -;
6980 -;
6990 -;*** menuepkt.invert/normalis.***
7000 -mnormal   lda #0          ;normalisieren:
7010 -          .by $2c         ;flag=0
7030 -minvert   lda #1          ;invertieren:
7040 -          sta flag        ;flag=1
7050 -mnor      ldx #0
7060 -          stx line        ;invertierparameter
7070 -          inx             ;uebergeben (zeile,
7080 -          stx laenge      ;laenge,flag sind fest)
7090 -;
7100 -          ldx mcount      ;startspaalte und
7110 -          lda start,x     ;invertierbreite
7120 -          sta col         ;bestimmen
7130 -          lda length,x    ;und invertierroutine
7140 -          sta breite      ;aufrufen
7150 -;
7160 -          jmp invjsr      ;rts !!!
7170 -;
7180 -;
7190 -;*** menue ausgeben ***
7200 -mprint    jsr wparam      ;pointer auf 1.menue-
7210 -          lda wpoilow,x   ;string uebergeben und
7220 -          ldy wpoihigh,x  ;malroutine aufrufen
7230 -          jmp winjsr      ;rts !!!
7240 -;
7250 -;
7260 -;*** untergrund retten/holen ***
7270 -liesgr    lda #0
7280 -          .by $2c
7300 -holgr     lda #1
7310 -          sta flag
7320 -lies1     lda #0          ;aus/in puffer nr.0
7330 -          sta puffer      ;holen/schreiben
7340 -          jsr wparam      ;windowparameter holen
7350 -          jmp pufferjsr   ;einsprung + rts !!!
7360 -;
7370 -;
7380 -;*** windowparam. uebergeben ***
7390 -wparam    ldx mcount      ;ueber 'mcount'
7400 -          lda start,x     ;(=aktuelle menuenr.)
7410 -          sta col         ;werden die parameter
7420 -          lda #1          ;des aktuellen
7430 -          sta line        ;windows aus den
7440 -          lda wbreite,x   ;tabellen ermittelt
7450 -          clc             ;(col,line,breite,
7460 -          adc #2          ;laenge)
7470 -          sta breite
7480 -          lda wlaenge,x
7490 -          clc
7500 -          adc #2
7510 -          sta laenge
7520 -          rts
```

Bild 16. Unterprogramme, die vom Hauptprogramm benötigt werden

Jene Leser, die sich die Mühe gemacht haben, den Programmablauf anhand des Source-Textes und meiner Erläuterungen zum Ablauf zu studieren, werden auch ohne »Bedienungsanleitung« zurechtkommen.

Anleitung zum vorgestellten Programm

Da dieses Programm jedoch von allen Lesern genutzt werden sollte, nicht nur von Assembler-Profis, folgt nun eine kurzgefaßte Anleitung zur Einbindung der Routinen in Basic-Programme.

Am Kapitelende finden Sie das MSE-Listing des Programms (Listing 2). Geben Sie das Programm ein und speichern mit einem Monitor den Bereich $C600 bis $CAFF, zum Beispiel unter dem Namen »PULL-DOWN«.

Laden Sie dieses »Objektprogramm« immer vor (!) dem Basic-Programm mit LOAD"PULL-DOWN",8,1:NEW. Wird Ihr Basic-Programm nicht compiliert, ist auch ein Nachladen des Maschinenprogramms (von Diskette) möglich. Die erste Zeile Ihres Basic-Programms sollte in diesem Fall lauten:

IF A=0 THEN A=1:LOAD"PULL-DOWN",8,1.

Pull-down-Menüs

Die Menüleiste und die einzelnen Kommandos der Untermenüs werden im Basic-Programm in Form von Stringvariablen definiert. Beachten Sie folgende Hinweise:

1. Die Menüleiste, eine beliebige Stringvariable, enthält die Namen aller Untermenüs, jeweils durch ein Leerzeichen getrennt.

2. Die Menükommandos werden in einem Stringarray definiert. Zuerst wird ein Leerstring angelegt, dessen Länge der maximalen Kommandolänge im ersten Menü entspricht. Die folgenden Strings enthalten die Kommandos des ersten Untermenüs.

3. Alle weiteren Untermenüs werden auf die gleiche Weise im selben (!) Stringarray definiert.

4. Dem letzten Kommando (=String) des letzten Menüs folgt ein Leerstring, der aus einem Leerzeichen (Space) besteht.

5. Die Routine wird aufgerufen mit SYS 50697,menü$, array$.

6. Mit den Cursor-Tasten können Sie Kommandos selektieren – und anschließend mit <RETURN> anwählen – und in den Untermenüs »blättern«.

7. Mit <CTRL> und dem jeweiligen Anfangsbuchstaben des Menünamens kann ein Untermenü direkt aktiviert werden. Mit dem jeweiligen Anfangsbuchstaben kann in einem aktiven Untermenü ein Kommando direkt angewählt werden.

8. Nach der Rückkehr aus der Routine befindet sich in Speicherzelle 167 die Nummer des aktiven Untermenüs und in 168 die Nummer des selektierten Kommandos. Die Numerierung beginnt in beiden Fällen mit null.

Zum Thema Pull-down-Menüs betrachten Sie bitte das Listing des ersten Demoprogramms (»Demoprogramm 1«).

Einzelne Routinen

Außer dem Gesamtprogramm können drei Unterprogramme recht sinnvoll eingesetzt werden. Allen drei Unterprogrammen müssen beim Aufruf die Parameter eines Rechtecks in folgender Form angegeben werden:

spalte/zeile: Die Bildschirmkoordinaten der oberen linken Ecke des Rechtecks (0/0 bis 39/24).
breite: Die Breite des Rechtecks in Spalten.
länge: Die Länge des Rechtecks in Zeilen.

1. Die Invertier-Routine

Diese Routine invertiert (flag=1) oder normalisiert (flag=0) einen beliebigen rechteckigen Bildschirmausschnitt.

Aufruf: SYS 50688+6,spalte,zeile,breite,länge, flag

Beispiel: SYS 50688+6,5,10,20,3,1 invertiert ein Rechteck, dessen linke obere Ecke sich an Position 5/10 (Spalte/Zeile) befindet, dessen Breite 20 Spalten beträgt und das drei Zeilen lang ist.

2. Die Mal-Routine

Diese Routine zeichnet einen Rahmen und füllt diesen mit Strings des angegebenen Arrays.

Aufruf: SYS 50688,spalte,zeile,breite,länge, array$

Beispiel: SYS 50688,2,4,10,5,A$(1) zeichnet einen Rahmen mit der linken oberen Ecke 2/4, der Breite zehn Spalten und einer Länge von fünf Zeilen. Die drei Innenzeilen des Rahmens werden mit dem Inhalt der Strings A$(1), A$(2) und A$(3) gefüllt.

3. Die Window-Routine

Die Window-Routine ermöglicht in Verbindung mit der Mal-Routine absolut professionelles »Windowing«. Mit dieser Routine kann ein beliebiger rechteckiger Bildschirmausschnitt in einen »Puffer« gerettet werden, bevor er durch das Window selbst überschrieben wird. Soll das Window wieder »ausgeblendet« werden, kann mit der gleichen Routine der ursprüngliche Bildschirminhalt aus dem Puffer auf den Bildschirm zurückgeschrieben werden.

Aufruf: SYS 50688+3,spalte,zeile,breite, länge,flag,puffer

flag: Mit diesem Parameter wird angegeben, ob ein Bildschirmausschnitt in einen Puffer kopiert werden soll, oder aber umgekehrt der Pufferinhalt auf den Bildschirm zu schreiben ist.

puffer: Sie können mehrere Windows, die sich gegenseitig teilweise verdecken (überlagern), beliebig ein- und ausblenden. In diesem Fall werden jedoch verschiedene Puffer benötigt, um jeden Window-Untergrund separat zu speichern. Mit dem Parameter »puffer« geben Sie an, in welchen Puffer ein Bildschirmauschnitt kopiert wird beziehungsweise welcher Puffer benutzt werden soll, um den ursprünglichen Bildschirminhalt wiederherzustellen.

Wichtig: Die Numerierung der Puffer beginnt mit null. Jeder folgende Puffer darf erst benutzt werden, wenn der vorige Puffer wenigstens einmal mit einem Inhalt gefüllt wurde. Das bedeutet: Verwenden Sie den zweiten Puffer Nummer Eins erst dann, wenn bereits durch einen vorhergehenden Aufruf der erste Puffer Nummer Null verwendet wurde und so weiter. Die Verwendung mehrerer Puffer ist nur dann nötig, wenn Sie mehrere Windows überlagern (siehe »Demoprogramm 2« Listing 3). Im »Normalfall« verwenden Sie bitte immer (!) Puffer Null.

Beispiel: SYS 50688+3,5,5,15,10,1,0 schreibt (flag=1) den Inhalt eines rechteckigen Bildschirmausschnitts mit der oberen linken Ecke 5/5, der Breite 15 Spalten und der Länge zehn Zeilen in Puffer Null. Mit SYS 50688+3,5,5,15,10,0,0 wird der ursprüngliche Inhalt dieses Bildschirmausschnitts wiederhergestellt (flag=0 => von Puffer nach Bildschirm kopieren).

Die Window-Routine ist zweifellos am schwierigsten anzuwenden, sie bietet jedoch enorme Möglichkeiten, wie vor allem Demoprogramm 2 zeigt. Beachten Sie außer dem bisher Gesagten, daß der Aufruf der Routine zum »Retten« beziehungsweise »Holen« eines Bildschirmausschnitts bis auf die Angabe »flag« identisch sein muß. (S. Baloui/ah)

```
10 REM *****************          <053>
20 REM * DEMOPROGRAMM 1 *         <191>
30 REM *****************          <073>
40 :                              <016>
50 :                              <026>
60 IF A=0 THEN A=1:LOAD"PULL-DOWN-OBJECT",
   8,1:REM ROUTINEN NACHLADEN     <091>
70 :                              <046>
100 DIM A$(40):PRINT CHR$(147)    <174>
110 K$="FILE DISK EDIT HELP"      <052>
120 A$(1)="{6SPACE}"              <079>
130 A$(2)="LOAD"                  <156>
140 A$(3)="SAVE"                  <084>
150 A$(4)="COPY"                  <167>
160 A$(5)="RENAME"                <253>
170 A$(6)="DELETE"                <028>
180 A$(7)="{10SPACE}"             <075>
190 A$(8)="DIRECTORY"             <117>
200 A$(9)="NAME"                  <255>
210 A$(10)="ID"                   <159>
220 A$(11)="COPY"                 <191>
230 A$(12)="DELETE"               <074>
240 A$(13)="VALIDATE"             <123>
250 A$(14)="INITIALISE"           <125>
260 A$(15)="{12SPACE}"            <051>
270 A$(16)="MOVE BLOCK"           <122>
280 A$(17)="COPY BLOCK"           <191>
290 A$(18)="DELETE BLOCK"         <017>
300 A$(19)="SAVE BLOCK"           <001>
310 A$(20)="LOAD BLOCK"           <204>
320 A$(21)="PRINT BLOCK"          <223>
330 A$(22)="{14SPACE}"            <216>
340 A$(23)="NOTEPAD"              <118>
350 A$(24)="CALCULATOR"           <125>
360 A$(25)="DATABASE"             <076>
370 A$(26)="WORD PROCESSOR"       <129>
380 A$(27)="SPREADSHEET"          <124>
390 A$(28)="FILE-HANDLING"        <219>
400 A$(29)="DISK-HANDLING"        <079>
410 A$(30)="{2SPACE}"             <200>
500 PRINT CHR$(19)                <036>
```

Listing 1. »Demoprogramm 1«

```
504 PRINT:PRINT"DIES IST EIN TEST DER ROUT
    INEN"                                 <013>
505 PRINT"ZUR VERWALTUNG VON PULL-DOWN-MEN
    UES"                                  <023>
510 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PR
    INT:PRINT:PRINT:PRINT:PRINT:PRINT     <129>
520 PRINT "AUFRUF : CTRL-TASTE DRUECKEN"  <063>
521 PRINT                                 <115>
522 PRINT"STEUERUNG: CURSORTASTEN ZUM BLAE
    TTERN"                                <247>
523 PRINT"VON MENUE ZU MENUE UND ZUR AUSWA
    HL IM"                                <023>
524 PRINT"UNTERMENUE. ANWAHL: 'RETURN'"   <091>
525 PRINT                                 <119>
526 PRINT"ALTERNATIV:'CTRL'+MENUEANFANGSBU
    CHSTABE"                              <083>
527 PRINT"UND MENUEPUNKTAUSWAHL DURCH"    <089>
528 PRINT"JEWEILIGEN ANFANGSBUCHSTABEN";  <196>
530 IF PEEK(653)<>4 THEN 530              <131>
540 SYS 50697,K$,A$(1)                    <011>
550 PRINT CHR$(19)                        <088>
560 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PR
    INT:PRINT:PRINT:PRINT:PRINT           <255>
565 PRINT"GETROFFENE AUSWAHL:"            <137>
570 PRINT "MENUE NR.{4SPACE,5LEFT}"PEEK(16
    7)+1                                  <061>
580 PRINT "MENUEPUNKT NR.{4SPACE,4LEFT}"PE
    EK(168)+1"                            <232>
590 GOTO 500                              <042>
```

Listing 1. »Demoprogramm 1«

```
Name : pull-down-object  c600 ca29
-----------------------------------
c600 : 4c 4a c6 4c c4 c6 4c 63   27
c608 : c7 4c a1 c7 8d 34 03 a2   22
c610 : 00 8e 35 03 20 fd ae 20   f2
c618 : 9e b7 8a ae 35 03 9d 36   59
c620 : 03 e8 ec 34 03 d0 ea 60   7c
c628 : a0 02 b1 a9 99 ad 00 88   83
c630 : 10 f8 60 a5 a9 18 69 03   90
c638 : 85 a9 90 02 e6 aa 60 ad   97
c640 : 37 03 18 6d 39 03 8d 39   01
c648 : 03 60 a9 04 20 0c c6 20   24
c650 : fd ae 20 8b b0 85 a9 84   05
c658 : aa 20 3f c6 ce 39 03 ce   1b
c660 : 38 03 ce 38 03 a9 00 8d   6d
c668 : 3a 03 20 87 c6 ee 3a 03   f0
c670 : 20 28 c6 20 87 c6 20 33   f0
c678 : c6 ad 37 03 cd 39 03 d0   97
c680 : ef ee 3a 03 4c 87 c6 ae   4f
c688 : 37 03 ac 36 03 18 20 f0   86
c690 : ff ae 3a 03 bd bb c6 20   ea
c698 : d2 ff a0 00 bd be c6 e0   41
c6a0 : 01 d0 06 c4 ad b0 02 b1   ef
c6a8 : ae 20 d2 ff c8 cc 38 03   f5
c6b0 : d0 ea ee 37 03 bd c1 c6   4b
c6b8 : 4c d2 ff b0 7d ad 60 20   8a
c6c0 : 60 ae 7d bd a9 06 20 0c   f2
c6c8 : c6 a5 00 a6 f0 8d 93 02   03
c6d0 : 8e 94 02 ae 3b 03 bd 93   e9
c6d8 : 02 85 ab bd 94 02 85 ac   08
c6e0 : 20 3f c6 ae 37 03 ac 36   d2
c6e8 : 03 18 20 f0 ff 20 51 c7   f3
c6f0 : a9 34 85 01 20 5a c7 a5   74
c6f8 : d2 85 aa a5 d1 18 6d 36   ec
c700 : 03 85 a9 90 02 e6 aa ac   9e
c708 : 38 03 88 ad 3a 03 d0 06   a5
c710 : b1 a9 91 ab d0 04 b1 ab   bb
c718 : 91 a9 88 10 ee a5 ab 18   9d
c720 : 6d 38 03 85 ab 90 02 e6   30
c728 : ac 20 51 c7 a9 37 85 01   9e
c730 : 20 5a c7 a9 11 20 d2 ff   02
c738 : a5 d6 cd 39 03 d0 ae ae   b2
c740 : 3b 03 e8 8a 0a aa a5 ab   6c
c748 : 9d 93 02 a5 ac 9d 94 02   f2
c750 : 60 ad 0e dc 29 fe 8d 0e   83
c758 : dc 60 ad 0e dc 09 01 8d   c7
c760 : 0e dc 60 a9 05 20 0c c6   39
c768 : 20 3f c6 ce 38 03 ae 37   78
c770 : 03 18 20 f0 ff a5 d1 18   4a
c778 : 6d 36 03 85 d1 90 02 e6   e9
c780 : d2 ac 38 03 b1 d1 ae 3a   f0
c788 : 03 f0 03 09 80 2c 29 7f   f2
c790 : 91 d1 88 10 ef ee 37 03   87
c798 : ae 37 03 ec 39 03 d0 d1   d3
c7a0 : 60 20 fd ae 20 8b b0 85   92
c7a8 : a9 84 aa 20 28 c6 a5 ad   ed
c7b0 : 48 a5 ae 48 a5 af 48 a0   ba
c7b8 : 00 a2 00 b1 ae c9 20 d0   9b
c7c0 : 03 c8 d0 f7 9d fd c9 98   7c
c7c8 : 9d eb c9 b1 ae c9 20 f0   9f
c7d0 : 05 c8 c4 ad 90 f5 98 38   ac
c7d8 : fd eb c9 9d f4 c9 e8 c8   c4
c7e0 : c4 ad 90 d7 8e 3f 03 20   c9
c7e8 : fd ae 20 8b b0 85 a9 84   9d
c7f0 : aa 20 28 c6 a2 ff d0 23   41
c7f8 : a9 00 8d 34 03 a0 00 b1   24
c800 : ae c9 20 f0 0b 20 33 c6   c5
c808 : 20 28 c6 ee 34 [illegible] [illegible] ed   46
c810 : c8 c4 ad 90 ea ad 34 03   ab
c818 : 9d 0f ca e8 a5 ad 9d 06   57
c820 : ca 20 33 c6 20 28 c6 a5   4a
c828 : a9 9d 18 ca a5 aa 9d 21   68
c830 : ca ec 3f 03 90 c2 a9 13   8c
c838 : 20 d2 ff 68 85 af 68 85   51
c840 : ae 68 85 ad a0 00 b1 ae   68
c848 : 20 d2 ff c8 c4 ad 90 f6   d4
c850 : a2 00 8e 3e 03 20 a8 c9   c5
c858 : 20 7c c9 20 9c c9 a9 ff   eb
c860 : 8d 40 03 20 e4 ff f0 fb   dc
c868 : 8d 41 03 a2 04 dd dc c9   e1
c870 : f0 05 ca 10 f8 30 12 8a   06
c878 : 0a aa bd e1 c9 8d 42 03   9b
c880 : bd e2 c9 8d 43 03 6c 42   55
c888 : 03 c9 40 b0 1c 09 40 ae   fe
c890 : 3f 03 ca dd fd c9 f0 05   bb
c898 : ca 10 f8 30 c6 8a 48 20   d1
c8a0 : ab c9 20 79 c9 68 aa 10   12
c8a8 : 62 ae 3e 03 bd 18 ca 85   24
c8b0 : a9 bd 21 ca 85 aa a9 00   2e
c8b8 : aa 20 28 c6 c8 b1 ae cd   c6
c8c0 : 41 03 f0 0f 20 33 c6 e8   29
c8c8 : 8a ae 3e 03 dd 0f ca 90   3c
c8d0 : e7 b0 90 8e 40 03 ad 3e   55
c8d8 : 03 85 a7 ad 40 03 85 a8   c1
c8e0 : 20 79 c9 4c ab c9 20 ab   9a
c8e8 : c9 20 79 c9 ae 3e 03 e8   14
c8f0 : ec 3f 03 90 16 a2 00 f0   a7
c8f8 : 12 20 ab c9 20 79 c9 ae   91
c900 : 3e 03 ca e0 ff d0 04 ae   82
c908 : 3f 03 ca 8e 3e 03 4c 55   25
c910 : c8 ad 40 03 30 03 20 4e   58
c918 : c9 ee 40 03 ae 3e 03 ad   0d
c920 : 40 03 dd 0f ca 90 05 a9   d4
c928 : 00 8d 40 03 20 51 c9 4c   ac
c930 : 63 c8 ad 40 03 30 f8 20   41
c938 : 4e c9 ad 40 03 d0 09 ae   16
c940 : 3e 03 bd 0f ca 8d 40 03   71
c948 : ce 40 03 4c 2c c9 a9 00   38
c950 : 2c a9 01 8d 3a 03 a9 02   a9
c958 : 18 6d 40 03 8d 37 03 ae   93
c960 : 3e 03 bd eb c9 8d 36 03   f4
c968 : ee 36 03 bd 06 ca 8d 38   47
c970 : 03 a9 01 8d 39 03 4c 68   e8
c978 : c7 a9 00 2c a9 01 8d 3a   e7
c980 : 03 a2 00 8e 37 03 e8 8e   f3
c988 : 39 03 ae 3e 03 bd eb c9   18
c990 : 8d 36 03 bd f4 c9 8d 38   f5
c998 : 03 4c 68 c7 20 bb c9 bd   57
c9a0 : 18 ca bc 21 ca 4c 55 c6   63
c9a8 : a9 00 2c a9 01 8d 3a 03   fd
c9b0 : a9 00 8d c4 c6 20 bb c9   45
c9b8 : 4c c9 c6 ae 3e 03 bd eb   3b
c9c0 : c9 8d 36 03 a9 01 8d 37   85
c9c8 : 03 bd 06 ca 18 69 02 8d   75
c9d0 : 38 03 bd 0f ca 18 69 02   f2
c9d8 : 8d 39 03 60 1d 9d 11 91   f5
c9e0 : 0d e6 c8 f9 c8 11 c9 32   72
c9e8 : c9 d6 c8 01 02 03 04 05   c1
c9f0 : 06 07 08 09 01 02 03 04   d1
c9f8 : 05 06 07 08 09 01 02 03   6a
ca00 : 04 05 06 07 08 09 01 02   ba
ca08 : 03 04 05 06 07 08 09 01   e6
ca10 : 02 03 04 05 06 07 08 09   00
ca18 : 01 02 03 04 05 06 07 08   08
ca20 : 09 01 02 03 04 05 06 07   19
ca28 : 08 ff ff ff ff ff ff ff   30
```

Listing 2. »pull-down-objekt« – Maschinencode zu den Menüroutinen. Das Programm wird von den Demoprogrammen 1 und 2 automatisch nachgeladen.

```
10 REM ******************                 <053>
20 REM * DEMOPROGRAMM 2 *                 <192>
30 REM ******************                 <073>
40 :                                      <016>
50 :                                      <026>
60 IF A=0 THEN A=1:LOAD"PULL-DOWN-OBJECT",
   8,1:REM ROUTINEN NACHLADEN             <091>
70 :                                      <046>
71 DIM A$(20),B$(20)                      <105>
72 FOR I=1 TO 20                          <091>
73 : A$(I)="DIES IST EIN TEST"            <244>
74 : B$(I)="EIN WEITERER TEST"            <027>
75 NEXT                                   <085>
76 :                                      <052>
100 PRINT CHR$(147)                       <129>
110 PRINT"'SPACE' SCHLIESST DAS JEWEILS OB
    ERSTE WINDOW":PRINT                   <212>
120 FOR I=1 TO 20                         <139>
130 : PRINT"{2SPACE}UEBERLAGERUNG MEHRERER
    WINDOWS"                              <134>
140 NEXT                                  <150>
150 SYS 50688+3,5,5,15,10,0,0             <221>
160 SYS 50688,5,5,15,10,A$(1)             <098>
170 SYS 50688+3,15,8,8,13,0,1             <093>
180 SYS 50688,15,8,8,13,B$(1)             <057>
200 GET A$:IF A$<>" " THEN 200            <175>
210 SYS 50688+3,15,8,8,13,1,1             <165>
220 GET A$:IF A$<>" " THEN 220            <196>
230 SYS 50688+3,5,5,15,10,1,0             <077>
```

Listing 3. »Demoprogramm 2«

Der Weg zum optimalen Programm

Wenn Sie Programme schreiben, sind bestimmte Vorgehensweisen erforderlich, um das Softwareprodukt übersichtlich und möglichst fehlerfrei zu erstellen. Wir zeigen Ihnen, wie das die Profis machen.

Um Mißverständnisse zu vermeiden: Dieser Artikel will Ihnen keine Programmiersprache beibringen, sondern allgemeine Prinzipien der Programmierung, die von der verwendeten Programmiersprache völlig unabhängig sind.

Diese Prinzipien sind vorwiegend für jene Leser interessant, die planen, größere »Programmpakete« zu erstellen. Solche Programme – zum Beispiel eine Textverarbeitung, Dateiverwaltung oder Buchhaltung – werden sehr schnell unübersichtlich. Daher ist eine eingehende Planung des »Programmsystems« notwendig, bevor (!) die Umsetzung (Codierung) in einer Programmiersprache erfolgt.

Vor allem im Heimcomputer-Bereich werden Programme ohne genaue Vorstellung erstellt. Die Vorgehensweise: Der Programmierer besitzt eine verschwommene Vorstellung vom endgültigen Programm, setzt sich an den Computer und »legt los«. Während der Programmierung stellt sich dann heraus, daß benötigte Programmteile vergessen wurden. Kein Problem, Programmzeilen lassen sich ja »einflicken«. Wenn das Programm – oftmals nach sehr kurzer Zeit – fertiggestellt ist, beginnen die eigentlichen Probleme:

- Verschiedene Programmfunktionen wurden einfach übersehen. Beispiel: »Hinterher« fällt dem Programmierer ein, daß in einer Dateiverwaltung »Datensätze« nicht nur eingetragen, geändert, gelöscht und gesucht werden, sondern es auch möglich sein sollte, die gesamte Datei nach einem beliebigen »Feld« des Datensatzes zu sortieren, nach dem Namen, der Postleitzahl oder dem Wohnort.
- Mangels Planung ist das gesamte Programm ungeeignet. Wieder das Beispiel Dateiverwaltung: Die Datensätze werden ungeordnet gespeichert. Soll die Datei sortiert ausgegeben werden, muß zuvor ein »Sortierlauf« durchgeführt werden, der die Datei zum Beispiel nach dem Feld »Name« ordnet und bei großen Datenmengen lange dauern kann. Interessiert Sie vorwiegend dieses Feld, wäre es besser, die Datensätze gleich beim Eintragen nach dem Namen geordnet zu speichern. In diesem Fall wäre die Datei ständig (!) – bezüglich des Felds »Name« – sortiert und Sortierläufe entfielen fast völlig.

Das eigentliche Problem besteht in der Programmänderung. Stellen Sie sich ein komplexes Programm vor, zum Beispiel die beschriebene Dateiverwaltung. Das Programm soll nun umgebaut werden, so daß Datensätze sofort beim Eintragen nach dem jeweiligen Namen geordnet in die Datei eingetragen werden.

Leider genügt es in den seltensten Fällen, den Programmteil zum Eintragen eines Satzes zu ändern (der sowieso komplett neu zu schreiben ist). Meistens sind die verschiedenen Teile eines Programms ineinander »verzahnt« und Änderungen in einem Teil machen Änderungen in allen damit zusammenhängenden Programmteilen notwendig. Bei großen Programmen sind Sie damit ebensolange beschäftigt wie mit der eigentlichen Programmerstellung.

Im geschilderten Beispiel wird der Arbeitsaufwand noch größer, da sich durch geordnete Speicherung auch der Programmablauf ändert. Wenn die Datei ständig sortiert ist, bietet es sich an, zusätzliche Funktionen einzubauen, zum Beispiel zum alphabetischen »Durchblättern« der Datei. Der Datensatz »Maier/Hamburg« befindet sich auf dem Bildschirm, und Sie wollen den alphabetisch folgenden Datensatz sehen (zum Beispiel »Meier/Mannheim«). Dank der geordneten Datei kann diese Funktion prinzipiell verwirklicht werden, jedoch bestimmt nicht problemlos, wenn vor der Erstellung des Programms noch niemand eine solche Funktion berücksichtigte.

Wenn sich mangels Planung wie in diesem Fall nachträglich die verwendeten »Datenstrukturen« ändern, ist das Programm oft reif für den Papierkorb und es ist leichter, das Programm neu zu schreiben, als das vorhandene Programm komplett umzubauen.

Noch ein Wort zum »Umbauen«. Das Umbauen und »Entfehlern« oder »Debuggen« von Programmen, die auf die beschriebene Weise entstanden sind, ist eine Sache für sich. Sie alle kennen bestimmt den Ausdruck »Spaghetti-Code«. Spaghetti-Programme entstehen vorwiegend, wenn »drauflosprogrammiert« wird. Während (!) der Programmerstellung wird der eigentliche Ablauf klarer und daraufhin fügt man hier schnell ein GOTO ein, dort ändert man eine Zeile, und an einer dritten Stelle werden gleich mehrere Programmzeilen eingefügt.

Das Ergebnis: Niemand, auch nicht der Programmierer selbst, versteht ein solches Programm. Änderungen sind extrem zeitaufwendig, wenn überhaupt machbar. Die Verzahnung der verschiedenen Programmteile führt oft dazu, daß Änderungen in einem Teil Fehler in einem anderen Teil nach sich ziehen, der zuvor einwandfrei funktioniert hat.

Das bisher Gesagte bezieht sich übrigens nur auf größere Programme. Theoretisch sollten auch Programme mit nur zehn Zeilen eingehend geplant werden, doch diese Vorgehensweise ist wohl unrealistisch.

Der Prozeß der Software-Entwicklung

Es existieren formale Richtlinien zur Beschreibung des Prozesses der Software-Entwicklung. An diese Richtlinien werde ich mich im folgenden anlehnen. Ich gehe davon aus, daß die wenigsten unter Ihnen planen, Programme im »Fremdauftrag« zu entwickeln, um zum Beispiel für einen Betrieb eine Buchhaltung oder Fakturierung zu erstellen. In diesem Fall entfallen natürlich Dinge wie Handbücher oder eingehende Sitzungen mit der Auftraggeber, um zu klären, welche Funktionen das Programm bieten soll.

Einige dieser Richtlinien sollten jedoch auch dann berücksichtigt werden, wenn Sie größere Programme nur für sich selbst schreiben. In den folgenden Erläuterungen werde ich als Beispiel die Erstellung eines Programms zur Verwaltung sogenannter »Pull-down-Menüs« verwenden, das in diesem Sonderheft abgedruckt ist. Sollten Sie nicht wissen, was ein Pull-down-Menü ist, schauen Sie sich bitte die Abbildungen in dem zugehörigen Artikel an.

Das Programm soll Pull-down-Menüs verwalten. Die

»Menüleiste« enthält die Namen aller »Untermenüs«. Der Benutzer steuert mit den Cursor-Tasten die Funktionsauswahl. Mit <CRSR>-rechts/-links wählt er das gewünschte Menü an, das unterhalb der Menüleiste über den Bildschirm gelegt wird.

Mit <CRSR>-oben/-unten wählt er das gewünschte Kommando im »aktiven« Menü aus (Bestätigung mit <RETURN>). Alternativ sollte eine »Direktanwahl« möglich sein. Gleichzeitiges Drücken von <CTRL> und der Taste, die dem Anfangsbuchstaben eines Menünamens entspricht, aktiviert das jeweilige Menü, beispielsweise aktiviert <CTRL+F> das Untermenü »File«.

Ähnlich sollte es möglich sein, in einem aktiven Untermenü ein Kommando direkt anzuwählen, indem der Anfangsbuchstabe des Kommandos eingegeben wird (mit <C> wird das Kommando »Copy« ausgewählt).

Wie erläutert, kann der Benutzer in den Untermenüs »blättern«, wobei diese Menüs »ein-« und wieder »ausgeblendet« werden. In jedem Fall muß nach dem Ausblenden eines Menüs der ursprüngliche Untergrund des »Menüwindows« wiederhergestellt werden, zum Beispiel ein Brief oder ein Datensatz.

Das Problem »Pull-down-Menüs verwalten« ist grob analysiert. Die prinzipielle Art und Weise des Programmablaufs ist festgelegt. Die Problemanalyse ist in diesem Fall recht einfach, da das Problem weniger komplex ist als zum Beispiel die Erstellung einer Dateiverwaltung.

Planung

In der Planungsphase wird geklärt, in welcher Programmiersprache der »Programmcode« erstellt wird und wie das Programm im einzelnen aufgebaut ist. Das Gesamtproblem wird in Teilprobleme gegliedert, zum Beispiel das Problem, den ursprünglichen Bildschirminhalt wieder herzustellen, wenn ein Untermenü »zugeklappt« wird. Diese Untergliederung führt dazu, daß das Gesamtprojekt überschaubar wird.

Als Programmiersprache wurde in diesem Fall der Geschwindigkeit wegen Assembler gewählt. Zugegeben: Sollten Sie ausschließlich Basic-Kenntnisse besitzen, entfällt für Sie dieser Teil der Planung.

Außerordentlich wichtig ist die Aufteilung des Gesamtprogramms in einzelne »Module«. Es gibt verschiedene Möglichkeiten zur Aufteilung. Am sinnvollsten erscheint die Unterteilung des Gesamtprogramms in »funktional« abgeschlossene Teile, das heißt in Teile, die eigenständig und möglichst unabhängig vom Rest des Programms sind. Ein Beispiel wäre eine Sortierroutine. Bei einer solchen Routine wird angegeben, welches Array sortiert werden soll. Ansonsten ist sie völlig unabhängig vom restlichen Programm.

Funktional abgeschlossene Module besitzen einen enormen Vorteil gegenüber anderen Arten der Aufteilung: Dank Ihrer Unabhängigkeit können sie in verschiedenen Programmen verwendet werden. Allmählich entsteht auf diese Weise eine »Modul-Bibliothek« mit einer Sortierroutine, einer Eingaberoutine und so weiter. Bei der Erstellung eines beliebigen Programms kann immer wieder auf diese Bibliothek zurückgegriffen werden. Mit der Aufteilung in funktionale Module ersparen Sie sich viel Arbeit, da bei späteren Programmen oftmals der »Griff in die Kiste« genügt, und schon besitzen Sie eine Sortierroutine, die sofort in das aktuelle Programm einzubauen ist.

Module besitzen folgende Kennzeichen:

1. Module besitzen meist die Form von Unterprogrammen, das heißt sie enden mit RETURN (Basic) oder RTS (Assembler).

2. Wichtig ist die exakte Definition der »Schnittstellen« des Moduls. Fast immer ist die Übergabe sogenannter »Parameter« notwendig, mit denen die Arbeitsweise des Moduls im jeweiligen Fall bestimmt wird. Ein Beispiel ist eine Sortierroutine, der anzugeben ist, wie groß das zu sortierende Array ist.

3. Schnittstellen existieren meist in beiden Richtungen: Das aufrufende Programm übergibt Parameter an das Modul, und das Modul übergibt Parameter an das aufrufende Programm, zum Beispiel das Ergebnis einer Berechnung.

Zur Verdeutlichung stelle ich eine kleine »Formatierroutine« vor, die zur formatierten Ausgabe von Zahlen in einer Tabelle verwendet wird:

```
460 REM **********************
470 REM * ZAHLENFORMATIERUNG *
480 REM **********************
490 :
500 REM AV=ANZAHL VORKOMMASTELLEN (PARAMETER HIN)
510 REM AN=ANZAHL NACHKOMMASTELLEN (PARAMETER HIN)
520 REM FZ=ZU FORMATIERENDE ZAHL (PARAMETER HIN)
530 REM FZ$=FORMATIERTE ZAHL ALS STRING
    (PARAMETER ZURUECK)
540 :
550 REM BSP.: AV=5:AN=2:FZ=1.5:GOSUB 580 => FZ$=
    "    1.50"
560 REM AV=5:AN=2:FZ=123:GOSUB 580 => FZ$=
    "  123.00"
570 :
580 FZ$=STR$(FZ):REM ZAHL IN STRING WANDELN
590 IF FZ>10^AV OR AV>9 THEN RETURN:REM
    BEREICHSPRUEFUNG
600 FOR A=1 TO LEN(FZ$):REM POSITION EINES
    EVENTUELL ENT-
610 : IF MID$(FZ$,A,1)="." THEN 630:REM HALTENEN
    DEZIMAL-
620 NEXT:REM PUNKTES IN <A> MERKEN
630 IF A>LEN(FZ$) THEN FZ$=FZ$+".":REM DEZ.PUNKT
    ANHAENGEN
640 FZ$="          "+FZ$:REM 10 SPACES VOR STRING
650 FZ$=RIGHT$(FZ$,LEN(FZ$)-9+AV-A):REM
    VORKOMMATEIL
660 FZ$=FZ$+"0000000000":REM 10 NULLEN ANHAENGEN
670 FZ$=LEFT$(FZ$,AV+AN+1):REM NACHKOMMATEIL
680 RETURN
```

Diese Routine wurde bereits vor längerer Zeit erstellt und soll hier nicht diskutiert und im Detail besprochen werden. Entscheidend ist die Verwirklichung der besprochenen Prinzipien.

Die Beispiele in den Zeilen 550 und 560 zeigen die Funktionsweise der Routine. Das aufrufende Programm übergibt die Parameter »AV«, die maximale Anzahl der Vorkommastellen, »AN«, die gewünschte Anzahl der Nachkommastellen und die zu formatierende Zahl »FZ«.

Die Routine formatiert die Zahl in der gewünschten Weise und übergibt sie im String »FZ$« an das aufrufende Programm.

Es handelt sich um ein funktionales Modul, da es eine klar umrissene Funktion hat, die Formatierung einer Zahl und vom restlichen Programm völlig unabhängig ist. Dadurch kann die Routine in beliebigen Programmen eingesetzt werden.

Ein wichtiger Gesichtspunkt bei der »modularen Programmierung« ist die Flexibilität der Module. Es wäre problemlos möglich gewesen, die vorgestellte Routine so zu schreiben, daß Zahlen beispielsweise immer (!) mit fünf Vor- und zwei Nachkommastellen formatiert werden. Möglicherweise ist dies die einzige Art der Formatierung, die in dem zu erstellenden Programm benötigt wird.

Der Haken daran: In einem anderen Programm wird eventuell eine andere Formatierungsart benötigt. Sie besitzen nun jedoch kein allgemein verwendbares Modul, sondern eine spezifische Routine, die zum Einsatz in einem anderen Programm umzuschreiben ist.

Achten Sie bitte darauf, daß Module flexibel und relativ allgemein sein sollten. Erreicht wird diese Flexibilität durch den Einsatz von Parametern, die die genaue Arbeitsweise der Routine festlegen (»AV«, »AN«).

Flexibilität kann man übrigens auch übertreiben. Überspitzt ausgedrückt: Eine Routine, die entweder Datensätze verwaltet oder aber Briefe schreibt, ist kein Modul mehr, sondern die Zusammenfassung zweier Programme. Achten Sie immer darauf, daß Module nicht zu umfangreich werden, sondern – wie im Beispiel – überschaubar bleiben, um mit wenigen Parametern auszukommen. Den optimalen Kompromiß zwischen Flexibilität und Überschaubarkeit muß jeder für sich selbst finden, ein Patentrezept kenne ich nicht.

Überschaubare Module besitzen einen weiteren Vorteil. Wenn ein Modul wie die vorgestellte Formatierungsroutine erstellt und zur einwandfreien Funktionsfähigkeit gebracht wurde, entfällt jegliche spätere Fehlersuche in diesem Programmabschnitt. Voraussetzung ist natürlich, daß das Modul auch tatsächlich in sich abgeschlossen und vom restlichen Programm unabhängig ist.

Bei einem Fehler in einem Programm, das die Formatierungsroutine verwendet, kann dieses Modul bei der Fehlersuche ausgeklammert werden. Generell ersparen Sie sich durch »modulare Programmierung« eine Unmenge an Arbeit bei der leider immer notwendigen Fehlersuche.

Zurück zur Verwaltung von Pull-down-Menüs. In welche Module das Gesamtprogramm aufgeteilt wird, ist nicht unbedingt eindeutig und hängt vom jeweiligen Programmierer ab.

Folgende Module wären denkbar:

1. Ein Modul, das beliebige Bildschirmausschnitte invertiert und wieder normalisiert, die »Invertierroutine«. Dieses Modul wird für verschiedene Aufgaben eingesetzt, zum Invertieren eines selektierten Menükommandos und zum Invertieren des Namens vom aktiven Untermenü.

2. Ein Modul, das ein Untermenü zeichnet und einen Rahmen (mit den Grafikzeichen des C 64) darum legt, die »Malroutine«.

3. Ein drittes Modul, das einen beliebigen Bildschirmausschnitt irgendwo im Speicher »puffert« und aus dem Puffer wieder auf den Bildschirm zurückschreiben kann, die »Windowroutine«. Wie in der Problemanalyse besprochen, soll nach dem »Zuklappen« eines Untermenüs der ursprüngliche Bildschirminhalt (Text, Datensatz) wiederhergestellt werden. Daher wird diese Routine benötigt, um vor dem Malen eines Untermenüs den Bildschirminhalt zu »retten« und ihn nach dem Verschwinden des Menüs wieder zu »holen«.

Alle drei Module sollen sowohl Schnittstellen zu Basic- als auch zu Assembler-Programmen enthalten. Wie dieses Problem gelöst wurde, können Sie im erwähnten Artikel nachlesen.

Zur Parameterübergabe ist folgendes zu sagen: Alle drei Module »behandeln« rechteckige Bildschirmausschnitte. Die Invertierroutine invertiert/normalisiert einen rechteckigen Ausschnitt. Die Malroutine zeichnet ein rechteckiges Untermenü mit den jeweiligen Kommandos. Die Windowroutine rettet/holt einen beliebigen rechteckigen Bildschirmausschnitt, der durch ein Untermenü überschrieben wird.

Der Überschaubarkeit wegen ist es angebracht, allen drei Modulen die Rechteck-Parameter auf die gleiche Art und Weise zu übergeben, zum Beispiel in der Form:
- Position der oberen linken Ecke des Rechtecks.
- Breite des Rechtecks.
- Länge des Rechtecks.

Diese drei Parameter legen das jeweilige Rechteck eindeutig fest. Assembler-Programmierer werden feststellen, daß diese Parameter allen Routinen in den gleichen Speicherzellen übergeben werden.

Die wichtigsten Module sind nun festgelegt. Einzelheiten entnehmen Sie bitte dem erwähnten Artikel.

Die Programmaufteilung ist festgelegt. Es besteht aus den erwähnten – und einer Reihe weiterer kleinerer – Modulen und selbstverständlich dem Hauptprogramm, das die Module zur Steuerung des Gesamtablaufs einsetzt.

Sowohl die Module als auch das Hauptprogramm werden nun »strukturiert«, der genaue Aufbau dieser Teile wird festgelegt. Entweder mit der Methode »Pi-mal-Daumen« und entsprechenden späteren Korrekturen, oder aber (empfehlenswerter) mit Hilfsmitteln zur Strukturierung wie Programmablaufplänen (PAP) oder Struktogrammen (Nassi/Shneidermann-Diagramm).

Verfeinerung

»Profis« verwenden meist Struktogramme, die zugegebenermaßen Vorteile bieten, jedoch »starrer« und unflexibler sind als PAPs.

Alle Hilfsmittel zur strukturierten Programmierung besitzen einen wesentlichen Vorteil: Sie zwingen (!) den Programmierer, seine verschwommenen Vorstellungen vom Programm- oder Modulaufbau zu spezifizieren. In allen Fällen wird der Programmablauf grafisch dargestellt, beim PAP zum Beispiel mit Symbolen wie Rechtecken, Rauten und Pfeilen.

Der Programmablauf kann nur dann mit diesen Symbolen grafisch dargestellt werden, wenn er dem Programmierer selbst klar ist. Diese Klärung tritt oft erst während der Erstellung von PAPs oder Struktogrammen ein. Ein PAP wird gezeichnet und wieder verworfen, der nächste PAP gemalt und so weiter.

Bei der Erstellung von PAPs und Struktogrammen wird die Methode der »schrittweise Verfeinerung« verwendet. Bild 1 zeigt einen »Grob-PAP« des Hauptprogramms. Zuerst ermittelt das Programm alle benötigten Parameter (Anzahl der Menüs, Menünamen, Menükommandos etc.). Auf einen »Menü-Initialisierungsteil«, der das aktuelle (nach dem Start das erste) Untermenü einblendet, folgt die »Hauptschleife«, zu der – fast – immer zurückgekehrt wird.

Zu Beginn der Hauptschleife wird die Tastatur abgefragt. Wurde eine Cursor-Taste gedrückt, verzweigt das Programm zum Abschnitt »Cursor-Taste ausführen«, der je nach Taste ein anderes Kommando selektiert (<CRSR>-oben/-unten), oder aber das aktive Untermenü »zuklappt« und ein benachbartes Menü aktiviert (<CRSR>-rechts/-links).

Wird ein Menü direkt angewählt (<CTRL> + Anfangsbuchstabe), verzweigt das Steuerungsprogramm zum Modul »Neues Menü aktivieren«. Die direkte Anwahl eines Kommandos über dessen Anfangsbuchstaben oder aber die Bestätigung von <RETURN> beendet das Gesamtprogramm, nachdem zuvor dem aufrufenden Programm mitgeteilt wird, welches Untermenü aktiv ist und welches Kommando dieses Menüs ausgewählt wurde.

Wurde kein Kommando endgültig ausgewählt, verzweigt das Programm zum Schleifenanfang, zur Abfrage der Tastatur. Eine Ausnahme bildet die Anwahl eines anderen Menüs, nach der zum Teil »Menü initialisieren« verzweigt wird.

Ausgehend von diesem Grob-PAP werden die einzelnen Programmbestandteile mit weiteren PAPs verfeinert. Bild 2 zeigt ein PAP, das den Ablauf des Moduls »Cursor-Taste ausführen« darstellt.

Wurde <CRSR>-rechts/-links gedrückt, werden folgende Aktionen ausgeführt:
- Der gerettete Untergrund des aktiven Menüs wird wiederhergestellt.
- Der invertierte Menüname wird wieder normalisiert.
- Der »Menüzähler«, der angibt, welches Menü aktiv ist, wird um eins erhöht (<CRSR>-rechts) beziehungsweise vermindert (<CRSR>-links).
- Der Menüzähler wird – wenn nötig – korrigiert. Enthält er

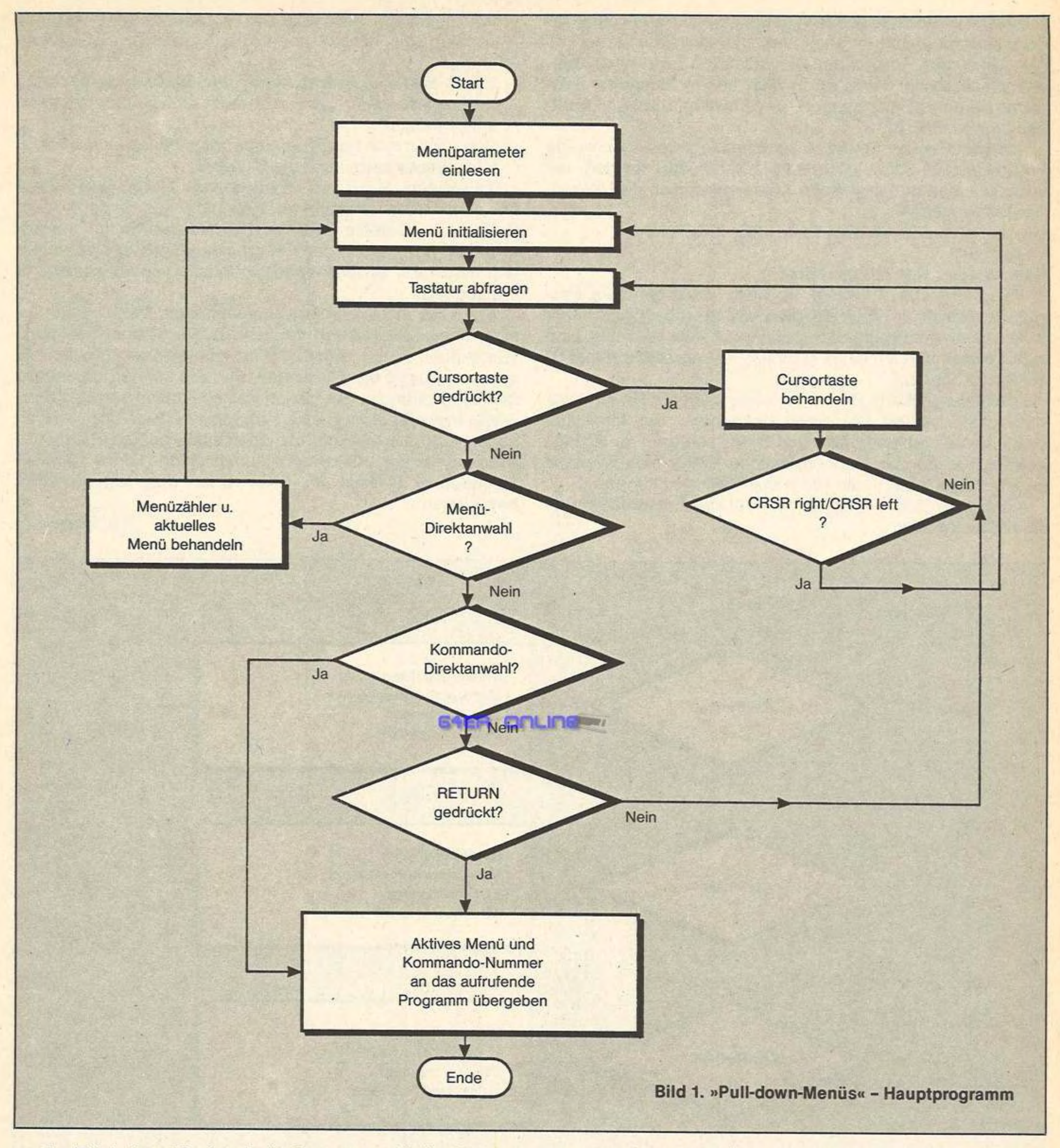


Bild 1. »Pull-down-Menüs« – Hauptprogramm

eine Zahl, die größer ist als die Gesamtanzahl aller Menüs, erhält er den Wert eins. »Weist« er vor das erste Menü, erhält er die Nummer des letzten Menüs.

Anschließend wird zum Programmteil »Menü initialisieren« verzweigt, das den Untergrund des neu selektierten Menüs rettet, dessen Namen in der Kommandozeile invertiert und das Menü selbst zeichnet.

<CRSR>-oben/-unten arbeitet ähnlich:

- Ein »Kommandozähler« wird um eins erhöht/vermindert.
- Das bisherige Kommando wird normalisiert.
- Der Kommandozähler wird korrigiert, wenn er größer als die Anzahl aller Kommandos des Menüs oder aber kleiner als die Nummer des ersten Kommandos ist.
- Das Kommando, auf das der Kommandozähler »zeigt«, wird invertiert.

Zuletzt wird zum Programmteil »Tastatur abfragen« verzweigt.

Die einzelnen Bestandteile eines solchen verfeinerten Ablaufplans können selbstverständlich wiederum verfeinert werden. Wie weit die Verfeinerung gehen sollte, muß der Programmierer entscheiden.

Nach der Strukturierung kann die eigentliche »Codierung« erfolgen, die Umsetzung der Ablaufpläne mit den Befehlen der gewählten Programmiersprache. Bis zu diesem Punkt ist der Prozeß der Programmentwicklung unabhängig von der Programmiersprache.

Da dieser Artikel weder einen Basic- noch einen Assembler-Kurs darstellt, wird auf die Umsetzung nicht näher eingegangen. Bei optimaler Programmplanung sollte dieser Teil am wenigsten Zeit in Anspruch nehmen.

Beachten Sie bitte, daß ein gut dokumentierter Programmcode äußerst wichtig ist. Mit Kommentaren sollten Sie auf keinen Fall sparen. Verwenden Sie zusätzlich Leer- oder Kommentarzeilen zur Trennung in sich abgeschlossener Programmteile und Einrückungen, um Schleifenführungen sichtbar zu machen.

Noch einen »Kommentar zu den Kommentaren«. Nicht jede Programmzeile sollte »sklavisch« kommentiert werden, um völlig sinnlose und ärgerliche Kommentare wie den folgenden zu vermeiden:

```
100 A=1:REM DER VARIABLEN A WIRD DER WERT 1
ZUGEWIESEN
```

Kein weiterer Kommentar nötig!

Ohne intensive Testläufe wird ein umfangreiches Programm niemals in jeder Situation wie gewünscht arbeiten. Selbst wenn der Programmierer keinen Fehler entdeckt, sind in der ersten Version eines komplexen Programms immer (!) Fehler vorhanden.

Empfehlung: Wenn Sie selbst keine weiteren Fehler mehr entdecken, setzen Sie einen Bekannten, der möglichst wenig von Computern, Bits und Bytes versteht, an Ihr Programm. Sie werden staunen, welche Fehler sich in einem anscheinend fehlerfreien Programm befinden können.

Außerordentlich wichtig ist der Test von Extremfällen und Spezialsituationen:

- Was passiert, wenn die Dateiverwaltung eine Adresse suchen soll, obwohl noch kein Datensatz eingegeben wurde?
- Stürzt das Programm ab, wenn die Sortierroutine ein Array mit nur einer oder gar keiner darin enthaltenen Variablen sortieren soll?
- Wie verhält sich das Programm, wenn der Speicher voll ist (Dateiverwaltung, Textverarbeitung)?

Verwenden Sie beim »Entwanzen« (Debuggen) Ihres Programms die vorhandenen Hilfsmittel des Systems, zum Beispiel die Befehle STOP und CONT des Basic-Interpreters, mit denen Sie das Programm gezielt unterbrechen und mit PRINT anschließend die Inhalte von Variablen prüfen können.

Wenn Sie in Assembler programmieren: Verschaffen Sie sich einen Monitor und verwenden Sie dessen Fähigkeit, »Breakpunkte« zu setzen. Wird ein solcher Breakpunkt erreicht, stoppt das Programm, und Sie können die Inhalte der Register und verwendeten Speicherzellen begutachten.

Viel mehr ist zum Thema Programmentwicklung nicht zu sagen. Detailliertere Informationen zum Thema strukturierte Programmierung oder Programmablaufpläne können Sie verschiedenen Artikeln im 64'er oder den Sonderheften entnehmen.

(S. Baloui/ah)

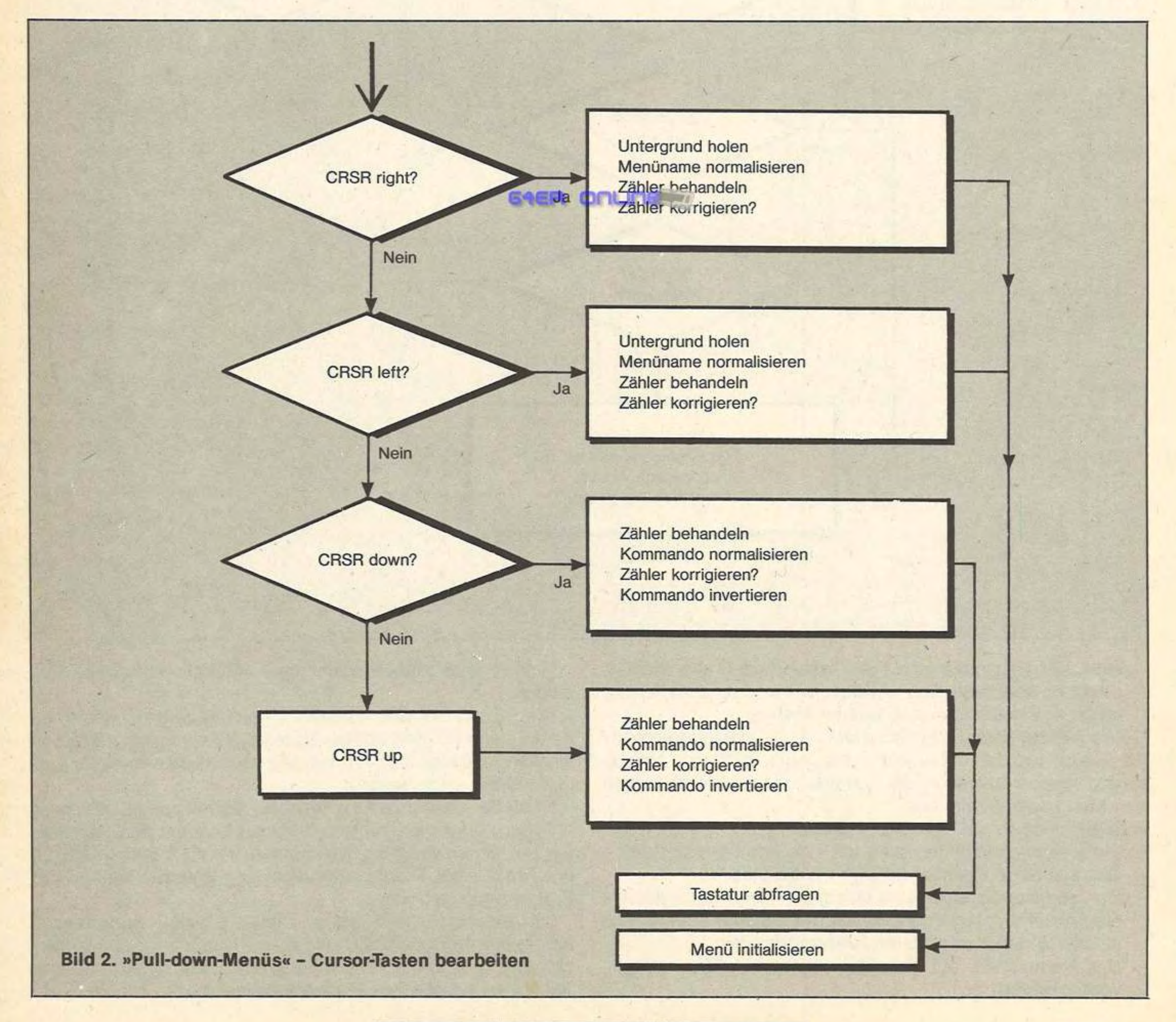
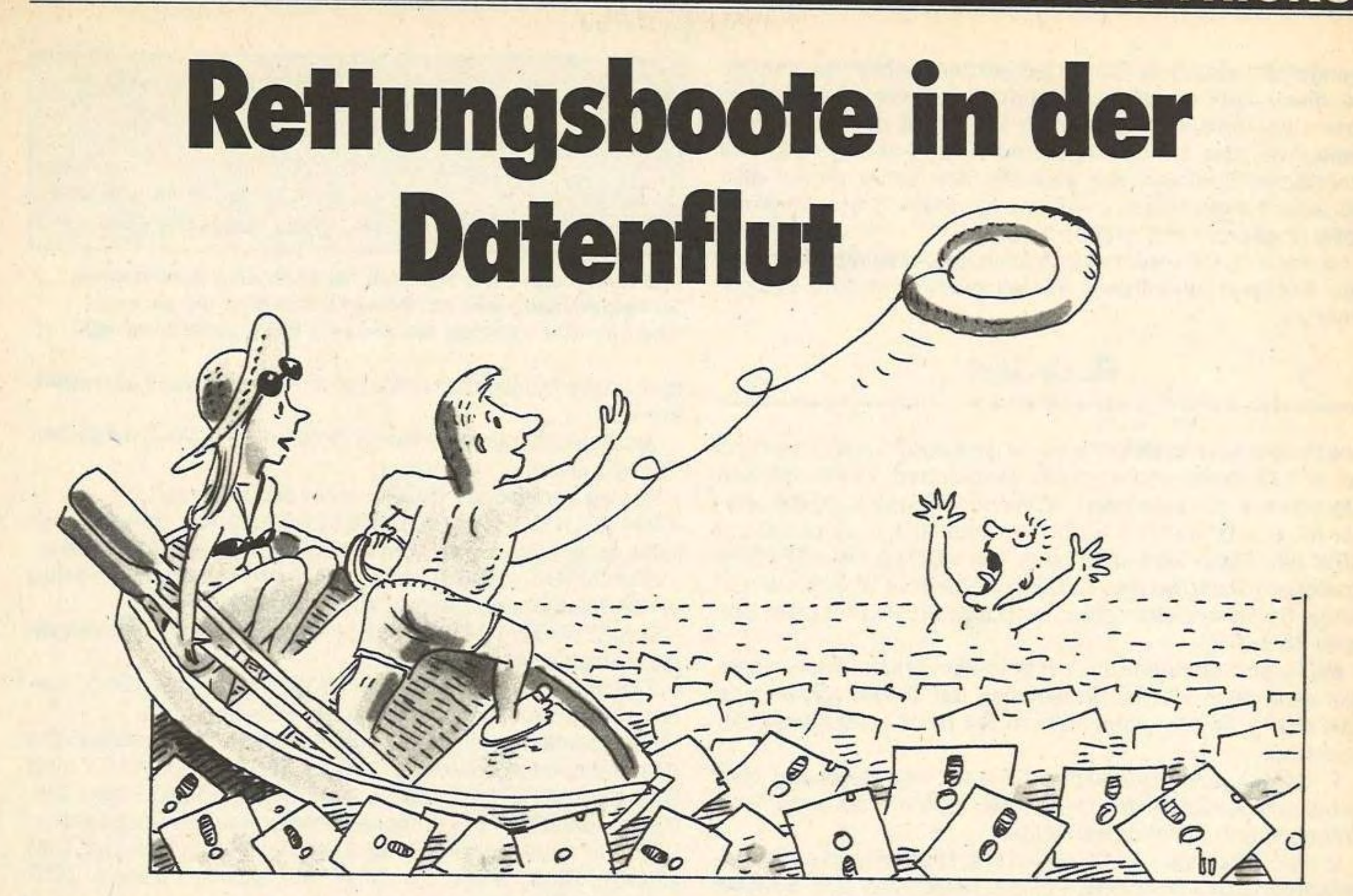


Bild 2. »Pull-down-Menüs« – Cursor-Tasten bearbeiten

Rettungsboote in der Datenflut



Verwalten Sie große Datenmengen, so kommen Sie um eine Sortierung dieser Daten kaum herum. Aber das Sortieren kostet in der Regel viel Zeit. Mit unseren neuen Sortierprogrammen wird das jedoch anders. Diese beiden Algorithmen sind die schnellsten und komfortabelsten, die wir je auf dem C 64 gesehen haben.

In diesem Artikel wollen wir Ihnen die beiden besten Sortierprogramme vorstellen, die wir bisher auf dem C 64 gesehen haben. Es handelt sich dabei um zwei Algorithmen, die beide in Assembler geschrieben sind. Der eine arbeitet dabei nach dem Prinzip des Shellsort-, während der andere den Quicksort-Algorithmus verwendet.

Bei beiden Programmen haben die Autoren darauf Wert gelegt, den entsprechenden Algorithmus sowohl schnell als auch komfortabel zu gestalten. Das ist ihnen auch gelungen, wie wir noch sehen werden.

Zuerst wollen wir uns den Shellsort-Algorithmus betrachten. Dazu eine kleine Wiederholung dessen, was wir schon in unserem Sortierkurs besprochen haben, der ab Ausgabe 4/1985 im 64'er abgedruckt war.

Shellsort (nach seinem Erfinder D. L. Shell 1959 benannt) ist eigentlich kein eigener Sortieralgorithmus. Er bedient sich vielmehr eines relativ einfachen Algorithmus (in unserem Beispiel dem »straight insertion«), wobei dieser einfache Algorithmus durch eine spezielle Optimierung in seiner Arbeit unterstützt wird.

Es zeigt sich leider immer wieder, daß die Sortierzeit bei einfachen Sortieralgorithmen im Verhältnis zur steigenden Anzahl von Elementen oft nahezu im Quadrat ansteigt. Das ist für den Anwender unzumutbar. Aus diesem Grund geht man bei Shellsort einen anderen Weg.

Man bedient sich hier eines einfachen Sortieralgorithmus, versucht jedoch die Anzahl der zu sortierenden Elemente möglichst klein zu halten. Das wird erreicht, indem das zu sortierende Feld in Teilfelder zerlegt wird, wobei sich die Elemente eines jeden Teilfeldes immer in einem bestimmten Abstand zueinander befinden. Diese Teilfelder werden nun einzeln von einem Unterprogramm, zum Beispiel straight insertion, sortiert. Nun wird der Abstand der Elemente der Teilfelder – wir nennen ihn Schrittweite – nach jedem Sortierdurchlauf halbiert, bis er schließlich 1 beträgt. Bei Schrittweite 1 erfolgt noch ein letzter Sortierdurchlauf und das Feld liegt fertig geordnet vor.

Anhand von Bild 1 läßt sich der gesamte Vorgang veranschaulichen.

Wir haben ein Ausgangsfeld, das mit zufällig gemischten Elementen (in unserem Fall die Zahlen von 0 bis 9) belegt ist. Als erste Schrittweite wird nun üblicherweise die halbe Anzahl der enthaltenen Elemente in dem Feld verwendet. In unserem Fall sind das fünf (bei ungeraden Zahlen wird bei der Teilung die Nachkommastelle abgeschnitten). Wir erhalten so fünf Teilfelder mit jeweils zwei Elementen. Jedes Element ist dabei die Schrittweite, also fünf Elemente, vom vorherigen entfernt. Das erste Teilfeld besteht deshalb aus dem ersten und dem sechsten Wert, das sind 9 und 7. Das zweite aus dem zweiten und dem siebten Wert, also 1 und 8, und so weiter.

So funktioniert Shellsort

Diese Teilfelder werden nun sortiert, ohne daß die Position innerhalb des Gesamtfeldes verändert wird. Sie können das in Bild 1 in der zweiten Zeile nach dem ersten Sortierdurchgang erkennen.

Nun wird die Schrittweite halbiert. Wir erhalten den Wert 2, was besagt, daß nur noch zwei Teilfelder mit jeweils fünf Elementen entstehen. Es gehören also das 1., 3., 5., 7. und 9. Element und das 2., 4., 6., 8. und 10. Element zusammen. Diese Teilfelder werden wiederum sortiert.

Dieses Halbieren der Schrittweite und Sortieren der Teilfelder geht bis zur Schrittweite 1.

In unserem Beispiel sind das drei Durchgänge. Es werden zwar auch hier immer mehr Elemente in einem Teilfeld zusam-

mengefaßt, so daß die Sortierzeit bei den Teilfeldern ansteigt. Da diese aber jeweils vom vorhergehenden Durchlauf in einem vorsortierten Zustand vorliegen, ist der zusätzliche Zeitaufwand bei steigender Elementanzahl sehr gering. Das mehrfache Sortieren der vorsortierten Felder dauert also insgesamt nicht so lang, wie das einmalige Sortieren eines völlig ungeordneten, großen Feldes.

Nun aber zu unserem ersten Assembler-Listing. Es enthält den Shellsort-Algorithmus in Verbindung mit dem straight insertion.

Flash-Sort

Das Programm in Listing 1 wird mit dem MSE eingegeben und auf eine Diskette oder Kassette gespeichert. Wollen Sie den Algorithmus nun einsetzen, so laden Sie das Programm wieder mit LOAD "FLASH-SORT",8,1 und tippen anschließend NEW ein. Flash-Sort steht dann von $CB20 bis $CFFE im Speicher (Startadresse = 52000). Darüber hinaus werden einige Speicherstellen in der Zeropage verwendet (zum Beispiel $FB-$FE).

Mit Flash-Sort haben Sie ein Sortierprogramm in der Hand, das wohl allen »Sortierproblemen« der Praxis gewachsen sein dürfte. Im einzelnen können Sie damit folgende Felder sortieren:

1. beliebige eindimensionale Felder; die Betonung liegt dabei auf »beliebig«, das heißt sowohl String- als auch Fest- (Integer-) und Fließkomma-Felder.
2. zweidimensionale String-Arrays. Hierbei wird eine »Tiefensortierung« vorgenommen, das heißt, sind zum Beispiel zwei Nachnamen gleich, dann wird nach Vornamen sortiert (eine genaue Erläuterung folgt später noch).
3. in obigen Fällen ist es auch möglich, nur einen Ausschnitt zu sortieren (Erstellung von Teillisten).
4. weiterhin kann in allen Fällen ein beliebiges eindimensionales Array mitsortiert werden.

Als erstes soll der wichtigste Fall besprochen werden und zwar das Sortieren eindimensionaler Arrays. Flash-Sort wird folgendermaßen aufgerufen:

1. SYS 52000, A$: Sortieren des String-Arrays A$.
2. SYS 52000, B, X%: Sortieren des Arrays B; in Abhängigkeit davon wird das Array X% mitsortiert (was damit gemeint ist, wird später erklärt).
3. SYS 52000 # 1, C%(von), C%(bis): nur den angegebenen Teilbereich sortieren. C%(bis) wird noch mitsortiert.
4. SYS 52000 # 1, D$(von), D$(bis), Y(von), Y(bis): Hier gilt das unter Punkt 2 und 3 Gesagte, bezogen auf das angegebene Teilfeld. Beachten Sie den Unterschied in der Syntax für ganze Felder und Teilfelder. Das Numerus-Zeichen (#) nach dem SYS-Befehl ist für Flash-Sort das Zeichen zum Sortieren eines Teilfeldes. Die folgende 1 ist das Zeichen zum Sortieren eines eindimensionalen Arrays.

Sicherheit steht hoch im Kurs

Bevor jetzt das Mitsortieren, beziehungsweise das Erstellen von Teillisten, erläutert werden soll, einige Bemerkungen zur Absturz- beziehungsweise Anwendungssicherheit von Flash-Sort. Diese Bemerkungen gelten entsprechend auch für das Sortieren zweidimensionaler String-Arrays.

Bevor Flash-Sort mit dem Sortieren beginnt, werden eine Reihe von Sicherheitsüberprüfungen durchgeführt. Im Normalfall werden Sie davon nichts merken, es sei denn, Sie erhalten die Fehlermeldung »FORMULA TOO COMPLEX ERROR«. Diese Meldung stammt von Flash-Sort. Sie haben dann in Ihren Sortier-Parametern irgendeinen (Flüchtigkeits-) Fehler begangen. Insbesondere bei der Sortierung von Teillisten ist die Möglichkeit relativ groß, irgend etwas zu »vergessen«.

9	1	3	0	4	7	8	6	5	2	**:Ausgangsfeld (zufällig gemischt)**
9	1	3	0	4	7	8	6	5	2	(9,7)(1,8)(3,6)(0,5)(4,2) = 5 Untereinheiten
7	1	3	0	2	9	8	6	5	4	(7,3,2,8,5)(1,0,9,6,4) = 2 Untereinheiten
2	0	3	1	5	4	7	6	8	9	(2,0,3,1,5,4,7,6,8,9) = 1 Untereinheit
0	1	2	3	4	5	6	7	8	9	:Ergebnis nach drei Durchläufen

Bild 1. So werden die Teillisten bei Shellsort während eines Sortierdurchlaufs erstellt. Bemerkenswert ist die geringe Anzahl an Durchläufen, bis das Feld fertig sortiert vorliegt.

Als Beispiel folgt eine kleine Zusammenstellung möglicher Fehlerquellen:

- Das zu sortierende Feld ist nicht dimensioniert.
- DIM A$(1000),A%(100):SYS 52000,A$,A%: unterschiedliche Anzahl von Elementen in den zu sortierenden Feldern.
- SYS 52000 # 1,A$(0),AA$(100): unterschiedliche Namen in der Teillistenangabe des zu sortierenden Feldes.
- SYS 52000 # 1,A$(1000),A$(100): Feldende ist kleiner als der Feldanfang.
- SYS52000 # 1,A$(100),A$(1000),B%(100),B(1000): unterschiedlicher Variablentyp in der Parameterliste.

Der Vollständigkeit halber seien noch die Fehlermeldungen des Basic-Interpreters (SYNTAX,ILLEGAL QUANTITY und BAD SUBSCRIPT ERROR) erwähnt. Diese Meldungen werden automatisch bei entsprechenden Fehlern ausgegeben und sind nicht spezifisch für Flash-Sort (zum Beispiel: DIM A$(10):PRINT A$(4711) führt natürlich zu einem BAD SUBSCRIPT ERROR).

Mitsortieren von Feldern

Sie werden sich vielleicht gefragt haben, was unter den Begriffen »Mitsortieren« und »Sortieren von Teilfeldern« zu verstehen ist. Wir möchten das am Beispiel einer Adreßverwaltung erklären. Nehmen Sie dazu folgenden Fall an:

Ihre Datensätze stehen in einer REL-Datei auf der Diskette. Die Nachnamen lesen Sie (als Indexfeld) in das String-Array NA$, die zugehörigen Datensatznummern in das Array RC% ein. Wenn Sie nun die Datensätze sortiert ausdrucken lassen wollen (SYS 52000,NA$,RC%), müssen die Record-Nummern natürlich bei den zugehörigen Namen »bleiben«. Das Feld der Record-Nummern muß also anhand der Namen mitsortiert werden, womit wir das Wort »mitsortieren« erklärt hätten.

In der Praxis bleibt jedoch für die Druckausgabe noch ein kleines Problem: Ihre Datensätze sind jetzt zwar nach Nachnamen geordnet, es kommt aber vor, daß zwei Leute den gleichen Nachnamen haben. Auf dem Papier sieht es dann etwas unschön aus, wenn »Schmidt Hans« vor »Schmidt Adam« zu finden ist. Dazu ein Vorschlag: Dimensionieren Sie für die Druckausgabe ein kleines Feld. In dieses Feld lesen Sie nun alle Datensätze ein, die mit dem gleichen Buchstaben (des Nachnamens) beginnen. Jetzt sortieren Sie einfach dieses Array »nach«. Wenn sich Ihre Datensätze vorher auch erst im letzten Datensatzfeld unterschieden haben, dann sind sie hinterher genau alphabetisch sortiert.

Jetzt zu dem Erstellen von Teillisten (Sortieren von Teil-Arrays). Sie könnten ja, um bei der Adreßdatei zu bleiben, Ihre Datei folgendermaßen aufgebaut haben:

Datensatznummern 1 bis 399: Adressen der Bekannten und Verwandten

Datensatznummern 400 bis 600: Adressen der Geschäftspartner

Sie können nun folgende Druckausgabe realisieren:

1. SYS 52000 # 1, NA$(1), NA$(399), RC%(1), RC%(399):

nur die Adressen der Bekannten/Verwandten sortieren

2. SYS 52000#1, NA$(400), NA$(600), RC%(400), RC%(600): nur die Adressen der Geschäftspartner sortieren

3. SYS 52000, NA$, RC%: alle Adressen sortieren.

Spätestens in Fall drei wird Ihnen noch etwas angenehm auffallen: Flash-Sort sortiert alle Leerstrings an das Ende eines (Teil-)Arrays.

Zwei Bemerkungen zum Sortieren von numerischen Arrays:

1. In einem speziellen Fall sortiert Flash-Sort etwas »eigenwillig« und zwar bei einem Integer-Array, wenn darin sowohl positive als auch negative Zahlen vorkommen. Hier werden die positiven Werte an den Anfang sortiert, gefolgt von den negativen Werten.

2. Allgemein kann es bei numerischen Zahlenfeldern sinnvoll sein, die Sortierreihenfolge umzudrehen, das heißt nicht das kleinste, sondern das größte Element an den Array-Anfang zu sortieren. Das ist mit Flash-Sort möglich. Die Sortierreihenfolge hängt von dem Wert einer einzigen Speicherstelle ab und zwar $CFFE (53246). Sie können den voreingestellten Wert mittels POKE ändern:

WERT = 1: kleinstes Element an den Anfang (voreingestellt)

WERT = 255: größtes Element an den Anfang

Im Gegensatz zum Diskettenlaufwerk ist bei der Datasette eine relative Datenspeicherung nicht möglich. Wenn Sie eine sequentielle Datei benutzen, müssen Sie diese zur Bearbeitung komplett in den Speicher des C64 einlesen. Auch hierzu wieder das Beispiel Adreßverwaltung.

Sie haben Ihre Datensätze jetzt (zwangsläufig) etwas anders organisiert :

Name 1: A$(1,0) Vorname1:
A$(1,1) Anschrift1:
A$(1,2) Bemerkung1:
A$(1,3) Name2:
A$(2,0) Vorname2:
A$(2,1) Anschrift2:
A$(2,2) Bemerkung2:
A$(2,3) Name3:
A$(3,0)...

Auch dieses Array kann Flash-Sort sortieren, allerdings in einer festgelegten Reihenfolge: In obigem Fall wird zuerst nach Namen sortiert; sind diese gleich, dann wird nach Vornamen sortiert. Sind diese auch gleich, dann nach der Anschrift, danach nach der Bemerkung. Diese Tiefensortierung geht also bis hin zum letzten Element der zweiten Dimension.

Sortieren zweidimensionaler String-Felder

Für das Sortieren ganzer zweidimensionaler String-Arrays hat sich am Aufruf nichts geändert (SYS 52000, Variablenname). Wenn Sie jedoch einen Teilbereich sortieren wollen, müssen Sie Flash-Sort folgendermaßen aufrufen:

SYS 52000#2, A$(von,0), A$(bis,0)

Beachten Sie dabei bitte folgende zwei Punkte:

- für die Anzahl der Elemente der zweiten Dimension müssen Sie immer jeweils die Zahl 0 eingeben, sonst erhalten Sie die Meldung »FORMULA TOO COMPLEX ERROR«.
- nach dem »#«-Zeichen steht eine 2; im Prinzip kann hier ein Wert zwischen 0 und 255 stehen, mit Ausnahme der 1. Sollten Sie hier doch eine 1 eingeben, so werden nur die Namen sortiert. Dadurch geraten Ihre Datensätze durcheinander und könnten etwa so aussehen:

Name 1: A$(5,0) Vorname1:
A$(1,1) Anschrift1:
A$(1,2) Bemerkung1:
A$(1,3) Name2:
A$(7,0) Vorname2:
A$(2,1) Anschrift2:
A$(2,2) Bemerkung2:
A$(2,3) ...

Auch beim Sortieren zweidimensionaler String-Arrays kann ein beliebiges eindimensionales Array mitsortiert werden. Wenn Sie eine andere Sortierreihenfolge wünschen, zum Beispiel erst nach Vornamem dann nach Namen sortieren wollen, müssen Sie vorher (zum Beispiel mit einer SWAP-Routine) alle Namen mit den Vornamen vertauschen. Solche Routinen sind im 64'er ja bereits des öfteren vorgestellt worden, so daß wir nicht näher darauf eingehen wollen.

Für alle Interessierten ist in Listing 2 der Quellcode von Flash-Sort abgedruckt. Diesen können Sie nach Belieben verändern, wenn Sie eigene Erweiterungen einbauen wollen. Zum Schluß dieser Anleitung ist eigentlich nur noch eine Frage offen: Warum heißt dieses Sortierprogramm Flash-Sort? Nun ja, es blinkt mit dem Bildschirmrahmen während des Sortiervorgangs.

Quicksort legt los

Bei unserem zweiten Sortierprogramm, das wir Ihnen hier vorstellen wollen, handelt es sich um einen Quicksort-Algorithmus. Über dieses Sortierverfahren wurde bereits so oft berichtet, daß wir an dieser Stelle nicht näher auf die Funktionsweise eingehen wollen. Wer sich dennoch dafür interessiert, der sei auf die Ausgabe 8/1985 des 64'er-Magazins verwiesen. Ab Seite 138 finden Sie alles Wissenswerte über Quicksort, das das schnellste derzeit bekannte Verfahren für das Sortieren von zufallsbesetzten Feldern ist.

Die folgende Assembler-Routine (Listing 3) unterscheidet sich von herkömmlichen Quicksort-Routinen nicht nur durch ihre Geschwindigkeit, sondern vor allem durch die erheblich höhere Flexibilität, mit der sich auch spezielle Sortierprobleme lösen lassen.

Die Routine befindet sich im Bereich $C000 bis $C316, benötigt während des Sortierlaufs jedoch zusätzlich die Bereiche $C400 bis $C700 zum Aufbau von Software-Stacks, die bei Quicksort wegen dessen rekursiver Arbeitsweise dringend notwendig sind, um den Hardware-Stack des Prozessors nicht zu überlasten.

Der Aufruf von Quicksort ist aufgrund der Flexibilität etwas umfangreich und lautet:

SYS 49152, ZEICHEN, FELD, SORT(X), SORT(Y), MITSORT(X), MITSORT(Y) wobei »MITSORT(X)« und »MITSORT(Y)« optionale Parameter sind, die nicht unbedingt angegeben werden müssen (nähere Erläuterung folgt).

Oftmals stellt sich bei der Arbeit mit einer Dateiverwaltung folgendes Problem: Die komplette (kleine) Datei befindet sich im Speicher des Computers in einem String-Array. Jeder String entspricht einem Datensatz, zum Beispiel einer Adresse:

A$(1)="Maier/Hans/Willistr.3/6800 Mannheim"
A$(2)="Bauer/Stefan/Friedrichstr.5/6700 Ludwigshafen"
A$(3)="Mueller/Gerhard/Aalweg 22/4000 Duesseldorf"

Herkömmliche Sortierroutinen versagen, wenn dieses Array nach einem Teil, zum Beispiel nach dem Namen, sortiert werden soll. Unser Quicksort gestattet Ihnen die Sortierung eines String-Arrays nach einem beliebigen Teil innerhalb eines Strings.

Die Vorgehensweise: Die einzelnen Teil-Strings (Name, Vorname etc.) müssen mit einem beliebig wählbaren Sonderzeichen voneinander getrennt werden, zum Beispiel mit dem hier verwendeten Schrägstrich »/«. Beim Aufruf der Routine müssen Sie als Parameter »ZEICHEN« den ASCII-Code des verwendeten Trennzeichens angeben (ASCII-Code von »/«:

47) und als Parameter »FELD« die Nummer jenes String-Teils, der als Sortierkriterium verwendet werden soll. Um die im Beispiel verwendeten Adressen nach dem Feld »NAME« zu sortieren, wird daher als Parameter »FELD« die Zahl 1 angegeben (»NAME« ist der erste Teil im String), zur Sortierung nach dem Vornamen wird eine 2 angegeben.

Ein Aufruf lautet also zum Beispiel:
SYS 49152,47,2,A$(1),A$(3)

wobei die drei Strings wie folgt sortiert werden:
A$(1)= "Mueller/Gerhard/Aalweg 22/4000 Duesseldorf"
A$(2)= "Maier/Hans/Willistr.3/6800 Mannheim"
A$(3)= "Bauer/Stefan/Friedrichstr.5/6700 Ludwigshafen"

Wichtig: Wenn Sie diese Option der Sortierung nach einem String-Teil nicht benötigen, geben Sie bitte für die beiden Parameter »ZEICHEN« und »FELD« jeweils eine Null ein (SYS 49152,0,0,A$(1),A$(3)).

Im vorhergehenden Beispiel wurden beim Aufruf die Sortiergrenzen angegeben, also die Parameter »SORT(X)« und »SORT(Y)«. Diese Sortiergrenzen haben den gleichen Effekt wie schon bei Flash-Sort. Sie gestatten das Sortieren von Teillisten in einem Feld, wenn nicht die Sortierung des Gesamtfeldes gewünscht wird. Wenn Sie ein mit A$(99) dimensioniertes Feld komplett sortieren wollen, geben Sie ein:
SYS 49152,0,0,A$(0),A$(99)

Die Parameterübergabe

Mit den Parametern »MITSORT(X)« und »MITSORT(Y)« können Sie ein beliebiges Feld angeben, das, analog zu Flash-Sort, mitsortiert wird. Die angegebenen Grenzen »X« und »Y« müssen dabei exakt jenen Grenzen entsprechen, die beim eigentlich zu sortierenden Feld angegeben wurden.

Auf unser Feld A$ angewendet, lautet der Aufruf der Routine also zum Beispiel:
SYS 49152,0,0,A$(1),A$(30),R%(1),R%(30)

Die Angabe des mitzusortierenden Arrays ist optional, das heißt sie kann (wie in den ersten Beispielen) völlig entfallen, wenn sie nicht benötigt wird (im Gegensatz zu den Angaben »ZEICHEN« und »FELD«, bei denen zumindest der »Scheinparameter« Null anzugeben ist).

Alle bisher beschriebenen Möglichkeiten können beliebig miteinander kombiniert und auf beliebige Array-Typen angewendet werden, also auf String-Arrays, Integer-Arrays und Fließkomma-Arrays.

Ein leistungsstarkes Sortierprogramm also, das wir Ihnen mit Quicksort vorgestellt haben. Das Source-Listing von Quicksort zeigt Listing 4. Wie Sie sehen, bieten sowohl Flash-Sort als auch Quicksort ähnliche Ausstattungsmerkmale an. Quicksort ist natürlich ein wenig schneller als Flash-Sort, dafür benötigt Flash-Sort weniger Speicherplatz und ist vom Programm her leichter zu verstehen.

In Sachen Geschwindigkeit können sich beide Sortierprogramme die »Hände reichen«. Haben Sie ein Basic-Programm, das mit Dateiverwaltung arbeitet, so ist es im Prinzip egal, welchen der beiden Algorithmen Sie einsetzen. Das ist Ihrem Geschmack und dem jeweiligen Anwendungsfall überlassen.

Die Sortierzeiten für 1000 Elemente liegen bei Quicksort unter zwei (!) Sekunden und Flash-Sort folgt in dichtem Abstand nach. Das sind Zeiten, die niemandem mehr weh tun und die wir sicher an dieser Stelle als das Nonplusultra auf dem C 64 bezeichnen können.

(W. Strunz/S. Baloui/ks)




```
Name : flash-sort            cb20 cfff
--------------------------------------
cb20 : a9 03 20 fb a3 20 b3 cd   78
cb28 : 8a 48 a2 03 b5 57 95 fb   44
cb30 : ca 10 f9 e8 ad 3e cf f0   8c
cb38 : 51 ad 3d cf 08 f0 06 8e   e6
cb40 : 3e cf 20 d7 cd 20 2e cf   9f
cb48 : 85 62 86 63 a2 90 38 20   dc
cb50 : 49 bc 20 0c bc ac 3e cf   4b
cb58 : f0 27 20 a2 b3 a5 61 20   66
cb60 : 12 bb 20 f7 b7 8c 3f cf   d3
cb68 : 8d 40 cf aa 98 18 65 57   ed
cb70 : a8 8a 65 58 28 f0 0d c5   8b
cb78 : fe 90 06 d0 0b c4 fd b0   8a
cb80 : 07 4c 1a ce 85 fe 84 fd   6c
cb88 : a2 00 86 14 20 79 00 f0   fe
cb90 : 22 20 cc cd 8a 85 14 4a   69
cb98 : b0 02 a9 ff 85 15 8a 0a   f3
cba0 : aa ca 86 6c a9 00 38 e5   26
cba8 : 14 85 6b 20 21 cf 68 48   20
cbb0 : 20 b0 ce a0 00 84 2a 68   8e
cbb8 : 85 6d c9 03 f0 07 ae 3e   45
cbc0 : cf d0 be f0 2f a5 fd a6   2a
cbc8 : fe 38 e9 03 b0 01 ca 85   06
cbd0 : 55 86 56 b1 55 d0 1a a5   c4
cbd8 : 59 38 e5 14 85 59 b0 02   33
cbe0 : c6 5a 86 fe a5 55 85 fd   6c
cbe8 : c5 fb d0 dd e4 fc d0 d9   c8
cbf0 : 60 a5 6d 88 84 6f 4a b0   de
cbf8 : 02 a9 ff 85 6e a5 6d 0a   5d
cc00 : aa ca a9 00 38 e5 6d 85   ed
cc08 : 69 86 6a ad 20 d0 48 46   3b
cc10 : 6a a5 69 6a 24 6e 50 03   f1
cc18 : 29 fe 18 90 02 e5 6e aa   37
cc20 : d0 05 68 8d 20 d0 60 85   54
cc28 : 69 ee 20 d0 18 a5 fb aa   1e
cc30 : 65 69 85 55 a5 fc a8 65   05
cc38 : 6a 85 56 a5 14 f0 42 46   0d
cc40 : 6c a5 6b 6a 24 15 50 03   d9
cc48 : 29 fe 18 90 02 e5 15 85   b7
cc50 : 6b a5 57 85 4b 85 28 a5   e1
cc58 : 58 d0 22 a5 26 a4 27 18   aa
cc60 : 65 6d aa 90 01 c8 c5 55   51
cc68 : d0 04 c4 56 f0 a1 a5 14   11
cc70 : f0 0f 18 65 28 85 28 85   f5
cc78 : 4b a5 29 69 00 85 29 85   e9
cc80 : 4c 8a 85 26 84 27 18 24   62
cc88 : 6f 70 0a 20 46 cf b0 cb   f3
cc90 : 18 a5 71 a4 72 85 22 65   12
cc98 : 69 aa 98 65 6a c5 fe 90   1b
cca0 : 06 d0 b8 e4 fd b0 b4 85   1c
cca8 : 25 84 23 86 24 85 72 86   ee
ccb0 : 71 a4 14 f0 16 a5 4b 85   5d
ccb8 : 61 65 6b 85 63 85 4b a5   32
ccc0 : 4c 85 62 65 6c 85 64 85   a4
ccc8 : 4c a0 00 b1 24 f0 c1 85   76
ccd0 : 5e c8 b1 24 85 5f c8 b1   5d
ccd8 : 24 85 60 b1 22 85 5d 88   e2
cce0 : b1 22 85 5c 88 b1 22 85   39
cce8 : 5b f0 66 c5 5e f0 04 90   ac
ccf0 : 02 a5 5e aa b1 5c d1 5f   b6
ccf8 : 90 97 d0 55 c8 ca d0 f4   43
cd00 : a4 5b c4 5e 90 8b d0 49   8a
cd08 : ae 3e cf f0 c0 86 2a 20   10
cd10 : 15 cf c6 2a f0 b7 20 f6   3f
cd18 : ce a0 02 b1 47 99 3b cd   b7
cd20 : b1 49 99 3e cd 88 d0 f3   f0
cd28 : b1 49 f0 a1 85 46 b1 47   ce
cd30 : f0 1f 85 45 c5 46 90 02   8f
cd38 : a5 46 aa b9 11 47 d9 15   bf
cd40 : 08 90 b5 d0 0c c8 ca d0   ec
cd48 : f2 a4 45 c4 46 90 a9 f0   e8
cd50 : c1 ac 3e cf f0 03 20 d9   4c
cd58 : ce a4 14 f0 20 88 b1 61   6b
cd60 : aa b1 63 91 61 8a 91 63   65
cd68 : 88 10 f3 c8 38 a5 61 85   4f
cd70 : 63 e5 6b 85 61 a5 62 85   29
cd78 : 64 e5 6c 85 62 a5 5b 91   7e
cd80 : 24 a5 5e 91 22 c8 a5 5c   f8
cd88 : 91 24 a5 5f 91 22 c8 a5   19
cd90 : 5d 91 24 a5 60 91 22 a5   da
cd98 : 22 a6 23 e4 27 d0 04 c5   07
cda0 : 26 f0 87 85 24 86 25 e5   a8
cda8 : 69 85 22 8a e5 6a 85 23   bc
cdb0 : 4c db cc a9 00 8d 3d cf   53
cdb8 : 8d 3e cf 20 79 00 c9 23   61
cdc0 : d0 0a 20 9b b7 ca 8e 3e   99
cdc8 : cf ce 3d cf 20 fd ae 20   35
cdd0 : 9e ad ac 3d cf d0 6f a5   a4
cdd8 : 2f a6 30 85 57 86 58 c5   ae
cde0 : 31 d0 04 e4 32 f0 33 a0   d0
cde8 : 00 b1 57 c8 c5 45 d0 04   82
cdf0 : b1 57 c5 46 08 c8 b1 57   c3
cdf8 : 18 65 57 85 59 c8 b1 57   9b
ce00 : 65 58 85 5a aa a5 59 28   cb
ce08 : d0 d1 c8 b1 57 c9 01 f0   d3
ce10 : 1c c9 02 d0 05 ad 3e cf   02
ce18 : f0 05 a2 19 6c 00 03 c8   bb
ce20 : b1 57 d0 f6 c8 b1 57 8d   22
ce28 : 3e cf a9 09 2c a9 07 18   36
ce30 : 65 57 85 57 90 02 e6 58   f2
ce38 : a2 05 a5 46 10 02 ca ca   61
ce40 : a5 45 10 01 ca 60 a0 03   e4
ce48 : b9 45 00 48 88 10 f9 20   de
ce50 : fd ae 20 9e ad 20 38 ce   db
ce58 : 8a 18 65 47 85 59 a4 48   77
ce60 : 90 01 c8 84 5a 68 c5 45   be
ce68 : d0 b0 68 c5 46 d0 ab 68   cd
ce70 : a8 68 c5 30 90 a4 d0 04   3d
ce78 : c4 2f 90 9e 85 58 84 57   a8
ce80 : c5 5a 90 c1 d0 94 c4 59   46
ce88 : b0 90 60 4a 29 02 08 a5   f0
ce90 : 22 0a aa a5 23 2a b0 f0   3f
ce98 : a8 28 90 13 f0 08 8a 0a   68
cea0 : aa 98 2a a8 b0 e2 18 8a   ce
cea8 : 65 22 aa 98 65 23 a8 60   af
ceb0 : 20 8b ce 86 24 84 25 a5   60
ceb8 : 69 85 22 a5 6a 85 23 a5   cc
cec0 : 14 20 8b ce c4 25 d0 a5   a5
cec8 : e4 24 d0 a1 a5 fb c5 57   27
ced0 : d0 06 a5 fc c5 58 f0 b0   f0
ced8 : 60 88 84 2a 20 15 cf 20   0d
cee0 : f6 ce a0 02 b1 47 aa b1   09
cee8 : 49 91 47 8a 91 49 88 10   c3
cef0 : f3 c6 2a d0 ea 60 a5 47   c2
cef8 : 18 6d 3f cf 85 47 a5 48   4a
cf00 : 6d 40 cf 85 48 18 a5 49   a0
cf08 : 6d 3f cf 85 49 a5 4a 6d   7f
cf10 : 40 cf 85 4a 60 a0 03 b9   6d
cf18 : 22 00 99 47 00 88 10 f7   fe
cf20 : 60 a5 fd 38 e5 fb 85 69   00
```

```
cf28 : a5 fe e5 fc 85 6a a5 59  5a
cf30 : 38 e5 57 85 22 aa a5 5a  a4
cf38 : e5 58 85 23 60 ea ea ea  ee
cf40 : ea a5 71 a4 72 18 85 5b  a2
cf48 : 84 5c 65 69 aa 98 65 6a  5b
cf50 : c5 fe 90 06 d0 e6 e4 fd  4d
cf58 : b0 e2 a8 85 72 86 71 85  80
cf60 : 5e 86 5d a5 14 f0 14 a5  72
cf68 : 4b 85 47 65 6b 85 4b 85  0f
cf70 : 49 a5 4c 85 48 65 6c 85  bc
cf78 : 4c 85 4a 8a 24 6e 70 60  a3
cf80 : 20 a2 bb a5 5b a4 5c 20  21
cf88 : 5b bc aa f0 b4 cd fe cf  5f
cf90 : f0 af a4 6d 88 b1 5b aa  07
cf98 : b1 5d 91 5b 8a 91 5d 88  83
cfa0 : 10 f3 a4 14 f0 05 e6 2a  7d
cfa8 : 20 ee ce a5 5b a4 5c c4  7d
cfb0 : 27 d0 04 c5 26 f0 8a 85  18
cfb8 : 5d 84 5e e5 69 aa 98 e5  c6
cfc0 : 6a a8 86 5b 84 5c a5 14  75
cfc8 : f0 11 a5 47 38 85 49 e5  34
cfd0 : 6b 85 47 a5 48 85 4a e5  2a
cfd8 : 6c 85 48 8a 24 6e 50 a7  b0
cfe0 : ae fe cf a0 00 b1 5b d1  b4
cfe8 : 5d 90 0d d0 07 c8 c0 02  a8
cff0 : d0 f3 f0 9c 8a 49 ff aa  d2
cff8 : 8a 30 97 4c 41 cf 01 00  a0
```

Listing 1. Das MSE-Listing für Flash-Sort. Bitte beachten Sie die Eingabehinweise auf Seite 92.

```
101:  CB20                         .OPT P4,00
102:  CB20                         *=   $CB20   ;SYS 52000
                             ;
                             ;----------------------------------------
                             ;--------- QUELLTEXT FLASHSORT ----------
                             ;----------------------------------------
                             ;
108:  CB20                   TERM     =    $AD9E
109:  CB20                   STEST    =    $A3FB
110:  CB20                   KOMMA    =    $AEFD
111:  CB20                   CHRGOT   =    $0079
112:  CB20                   TYPE     =    $6D
113:  CB20                   RUND     =    $6E
114:  CB20                   NUMFL    =    $6F
                             ;
                             ;------ SPEICHERSTELLEN ZEROPAGE --------
                             ;------ NUR EINDIMENSIONAL/STRING -------
                             ;
                             ;-------------- HAUPTARRAY --------------
                             ;
                             ;$FB/$FC = HAUPTARRAY ANFANG
                             ;$FD/$FE = HAUPTARRAY ENDE+1
                             ;$22/$23 = A-ELEMENT HAUPTARRAY
                             ;$71/$72 = B-ELEMENT
                             ;$24/$25 = B-ELEMENT(EINSORTIEREN)
                             ;$26/$27 = TEILFELD ANFANG
                             ;$55/$56 = ENDE+1 DES TEILFELDES
                             ;$5B-$5D = DESKRIPTOR A-ELEMENT
                             ;$5E-$60 = DESKRIPTOR B-ELEMENT
                             ;$69/$6A = SCHRITTWEITE
                             ;  $6D   = PLATZBEDARF(2,3 ODER 5)
                             ;  $6E   = RUNDUNGSBYTE BEIM
                             ;          HALBIEREN DER SCHRITT-
                             ;          WEITE;INTEGER BENOETIGT
                             ;          SONDERBEHANDLUNG
                             ;  $6F   = FLAG NUM./STRINGARRAY
                             ;          NUM.=0 STRING=$FF
                             ;
                             ;-------------- NEBENARRAY --------------
                             ;
                             ;$57/$58 = ANFANG
                             ;$59/$5A = ENDE+1
                             ;$61/$62 = A-ELEMENT
                             ;$4B/$4C = B-ELEMENT
                             ;$63/$64 = B-ELEMENT(EINSORTIEREN)
                             ;$28/$29 = TEILFELDBEGINN
                             ;          (TEILFELDENDE UNNOETIG)
                             ;$6B/$6C = SCHRITTWEITE
                             ;  $14   = PLATZBEDARF(2,3 ODER 5)
                             ;  $15   = RUNDUNGSBYTE
                             ;          WIE $6D/$6E HAUPTARRAY
                             ;---------------- START -----------------
                             ;
154:  CB20 A9 03                LDA  #3       ;STACK AUF
155:  CB22 20 FB A3             JSR  STEST    ;6 BYTE TESTEN
156:  CB25 20 B3 CD             JSR  ARRAY    ;1.ARRAY HOLEN
157:  CB28 8A                   TXA
158:  CB29 48                   PHA           ;TYP MERKEN
159:  CB2A A2 03                LDX  #3
160:  CB2C B5 57       SETAR    LDA  $57,X    ;ARRAYGRENZEN
161:  CB2E 95 FB                STA  $FB,X    ;HAUPTARRAY
162:  CB30 CA                   DEX
163:  CB31 10 F9                BPL  SETAR
164:  CB33 E8                   INX
165:  CB34 AD 3E CF             LDA  FLAG1    ;2 DIM.ARRAY
166:  CB37 F0 51                BEQ  SETAR1   ;NEIN
167:  CB39 AD 3D CF             LDA  FLAG     ;JA,TEST AUF TEIL-
168:  CB3C 08                   PHP           ;BEREICH;WENN JA
169:  CB3D F0 06                BEQ  NORMAL   ;ENDE UND ANZAHL
170:  CB3F 8E 3E CF             STX  FLAG1    ;ELEMENTE 2.DIM.
171:  CB42 20 D7 CD             JSR  ARR3     ;HOLEN
172:  CB45 20 2E CF NORMAL      JSR  PLATZ
173:  CB48 85 62                STA  $62      ;GES.PLATZ DIVID.
174:  CB4A 86 63                STX  $63      ;DURCH ANZAHL
175:  CB4C A2 90                LDX  #$90     ;ELEMENTE 2.DIM.
176:  CB4E 38                   SEC           ; = OFFSET AUF
177:  CB4F 20 49 BC             JSR  $BC49    ;DAS 1.ELEMENT
178:  CB52 20 0C BC             JSR  $BC0C    ;DER 2.DIMENSION
179:  CB55 AC 3E CF             LDY  FLAG1
180:  CB58 F0 27                BEQ  FEHLF
181:  CB5A 20 A2 B3             JSR  $B3A2
182:  CB5D A5 61                LDA  $61
183:  CB5F 20 12 BB             JSR  $BB12
184:  CB62 20 F7 B7             JSR  $B7F7
185:  CB65 8C 3F CF             STY  OL       ;OFFSET LOW
186:  CB68 8D 40 CF             STA  OH       ;OFFSET HIGH
187:  CB6B AA                   TAX
188:  CB6C 98                   TYA
189:  CB6D 18                   CLC
190:  CB6E 65 57                ADC  $57
191:  CB70 A8                   TAY
192:  CB71 8A                   TXA
193:  CB72 65 58                ADC  $58
194:  CB74 28                   PLP           ;NUR TEILARRAY
195:  CB75 F0 0D                BEQ  SETARA   ;NEIN
196:  CB77 C5 FE                CMP  $FE      ;TEST,OB ANGABE
197:  CB79 90 06                BCC  FEHLF    ;1.ELEMENT DER
198:  CB7B D0 0B                BNE  SETAR0   ;2.DIMENS. =0!
199:  CB7D C4 FD                CPY  $FD      ;WENN NEIN,FEHLER
200:  CB7F B0 07                BCS  SETAR0
201:  CB81 4C 1A CE FEHLF       JMP  FEHLER
202:  CB84 85 FE       SETARA   STA  $FE      ;GANZES ARRAY
203:  CB86 84 FD                STY  $FD      ;ENDE NEU SETZEN
                             ;----------------------------------------
205:  CB88 A2 00       SETAR0   LDX  #0       ;DEFAULT FUER
206:  CB8A 86 14       SETAR1   STX  $14      ;KEIN 2.ARRAY
207:  CB8C 20 79 00             JSR  CHRGOT
208:  CB8F F0 22                BEQ  DEL
209:  CB91 20 CC CD             JSR  ARR1     ;2.ARRAY HOLEN
210:  CB94 8A                   TXA           ;PLATZBEDARF NACH
211:  CB95 85 14                STA  $14      ;$14 UND
212:  CB97 4A                   LSR           ;RUNDUNGSBYTE
213:  CB98 B0 02                BCS  SETR     ;BERECHNEN
214:  CB9A A9 FF                LDA  #$FF     ;-INTEGER-
215:  CB9C 85 15       SETR     STA  $15      ;NEBENARRAY
216:  CB9E 8A                   TXA           ;STARTSCHRITTWEITE
217:  CB9F 0A                   ASL           ;AUF 511 ELEMENTE
218:  CBA0 AA                   TAX           ;HIGH=PLATZ*2-1
219:  CBA1 CA                   DEX
220:  CBA2 86 6C                STX  $6C
221:  CBA4 A9 00                LDA  #0       ;LOW=256-PLATZ
222:  CBA6 38                   SEC
223:  CBA7 E5 14                SBC  $14
224:  CBA9 85 6B                STA  $6B
225:  CBAB 20 21 CF             JSR  GPLATZ
226:  CBAE 68                   PLA           ;TEST AUF GLEICHE
227:  CBAF 48                   PHA           ;ANZAHL VON
228:  CBB0 20 B0 CE             JSR  TEST     ;ELEMENTEN
                             ;----------------------------------------
230:  CBB3 A0 00       DEL      LDY  #0       ;DEFAULTWERT
231:  CBB5 84 2A                STY  $2A      ;NUMER.ARRAY
232:  CBB7 68                   PLA           ;FUER MITSORTIEREN
233:  CBB8 85 6D                STA  TYPE
234:  CBBA C9 03                CMP  #3       ;NUMER.ARRAY
235:  CBBC F0 07                BEQ  DEL0     ;NEIN
236:  CBBE AE 3E CF             LDX  FLAG1    ;JA,TEST AUF
237:  CBC1 D0 BE                BNE  FEHLF    ;EINDIMENSIONAL
238:  CBC3 F0 2F                BEQ  START0
                             ;----------------------------------------
240:  CBC5 A5 FD       DEL0     LDA  $FD      :LEERSTRINGS
241:  CBC7 A6 FE                LDX  $FE      ;AM ENDE DES
242:  CBC9 38          DEL1     SEC           ;ARRAYS ENTFERNEN
243:  CBCA E9 03                SBC  #3       ;NUR WENN HAUPT-
244:  CBCC B0 01                BCS  DEL2     ;ARRAY STRING
245:  CBCE CA                   DEX
246:  CBCF 85 55       DEL2     STA  $55
247:  CBD1 86 56                STX  $56
248:  CBD3 B1 55                LDA  ($55),Y
249:  CBD5 D0 1A                BNE  START    ;KEIN LEERSTRING
250:  CBD7 A5 59                LDA  $59      ;NEBENARRAY
251:  CBD9 38                   SEC           ;VERKLEINERN
252:  CBDA E5 14                SBC  $14
253:  CBDC 85 59                STA  $59
254:  CBDE B0 02                BCS  DEL3
255:  CBE0 C6 5A                DEC  $5A
256:  CBE2 86 FE       DEL3     STX  $FE      ;HAUPTARRAY
257:  CBE4 A5 55                LDA  $55      ;VERKLEINERN
258:  CBE6 85 FD                STA  $FD
259:  CBE8 C5 FB                CMP  $FB      ;TEST OB ARRAY-
260:  CBEA D0 DD                BNE  DEL1     ;ANFANG SCHON
261:  CBEC E4 FC                CPX  $FC      ;ERREICHT IST
262:  CBEE D0 D9                BNE  DEL1     ;NEIN,WEITERMACHEN
                             ;
264:  CBF0 60                   RTS           ;****** ENDE ******
                             ;                 *NUR LEERSTRINGS *
                             ;------------ SORTIERBEGINN ------------
                             ;
268:  CBF1 A5 6D       START    LDA  TYPE     ;
269:  CBF3 88                   DEY           ;STRING=$FF
270:  CBF4 84 6F       START0   STY  NUMFL    ;NUM.=0
271:  CBF6 4A                   LSR           ;RUNDUNGSBYTE
272:  CBF7 B0 02                BCS  START1   ;BERECHNEN
273:  CBF9 A9 FF                LDA  #$FF     ;--INTEGER
274:  CBFB 85 6E       START1   STA  RUND
275:  CBFD A5 6D                LDA  TYPE
276:  CBFF 0A                   ASL           ;STARTSCHRITT-
277:  CC00 AA                   TAX           ;WEITE AUF 511
278:  CC01 CA                   DEX           ;ELEMENTE SETZEN
```

Listing 2. Der Source-Code zu Shellsort in Assembler

```
279:  CC02 A9 00            LDA  #0        ;(WIE OBEN)
280:  CC04 38               SEC
281:  CC05 E5 6D            SBC  TYPE
282:  CC07 85 69            STA  $69
283:  CC09 86 6A            STX  $6A
284:  CC0B AD 20 D0         LDA  53280     ;RAHMENFARBE
285:  CC0E 48               PHA            ;MERKEN
                      ;---------------------------------------
287:  CC0F 46 6A   BIG      LSR  $6A       ;SCHRITTWEITE
288:  CC11 A5 69            LDA  $69       ;HALBIEREN
289:  CC13 6A               ROR            ;START MIT 255 ELEM.
290:  CC14 24 6E            BIT  RUND      ;TEST AUF INTEGER
291:  CC16 50 03            BVC  BIGA      ;NEIN
292:  CC18 29 FE            AND  #$FE      ;JA,BIT 0 LOESCHEN
293:  CC1A 18               CLC            ;UND KEIN UEBERTRAG
294:  CC1B 90 02   BIGA     BCC  BIGB
295:  CC1D E5 6E            SBC  RUND
296:  CC1F AA      BIGB     TAX
297:  CC20 D0 05            BNE  BIG1
298:  CC22 68               PLA
299:  CC23 8D 20 D0         STA  53280
300:  CC26 60               RTS            ;****** ENDE ******
301:  CC27 85 69   BIG1     STA  $69
302:  CC29 EE 20 D0         INC  53280     ;BLINKEN
303:  CC2C 18               CLC
304:  CC2D A5 FB            LDA  $FB       ;TEILFELDSTART
305:  CC2F AA               TAX            ;=ARRAYANFANG
306:  CC30 65 69            ADC  $69       ;LOW IN X-REG
307:  CC32 85 55            STA  $55       ;HIGH IN Y-REG
308:  CC34 A5 FC            LDA  $FC       ;+SCHRITTW.
309:  CC36 A8               TAY            ;=TEILFELDENDE
310:  CC37 65 6A            ADC  $6A
311:  CC39 85 56            STA  $56
312:  CC3B A5 14            LDA  $14
313:  CC3D F0 42            BEQ  SET3
314:  CC3F 46 6C            LSR  $6C       ;SCHRITTWEITE
315:  CC41 A5 6B            LDA  $6B       ;NEBENARRAY
316:  CC43 6A               ROR            ;HALBIEREN
317:  CC44 24 15            BIT  $15       ;TEST AUF INTEGER
318:  CC46 50 03            BVC  BIG2      ;NEIN
319:  CC48 29 FE            AND  #$FE      ;JA,BIT 0 LOESCHEN
320:  CC4A 18               CLC            ;UND KEIN UEBERTRAG
321:  CC4B 90 02   BIG2     BCC  BIG3
322:  CC4D E5 15            SBC  $15
323:  CC4F 85 6B   BIG3     STA  $6B
324:  CC51 A5 57            LDA  $57
325:  CC53 85 4B            STA  $4B
326:  CC55 85 28            STA  $28
327:  CC57 A5 58            LDA  $58
328:  CC59 D0 22            BNE  SET2A     ;UNBEDINGTER SPRUNG
                      ;---------------------------------------
330:  CC5B A5 26   SET      LDA  $26       ;NAECHSTES TEILFELD
331:  CC5D A4 27            LDY  $27       ;BEARBEITEN
332:  CC5F 18               CLC
333:  CC60 65 6D            ADC  TYPE
334:  CC62 AA               TAX
335:  CC63 90 01            BCC  SET1
336:  CC65 C8               INY
337:  CC66 C5 55   SET1     CMP  $55       ;WENN ALLE TEIL-
338:  CC68 D0 04            BNE  SET2      ;FELDER SORTIERT,
339:  CC6A C4 56            CPY  $56       ;DANN SCHRITTWEITE
340:  CC6C F0 A1            BEQ  BIG       ;HALBIEREN
341:  CC6E A5 14   SET2     LDA  $14
342:  CC70 F0 0F            BEQ  SET3
343:  CC72 18               CLC
344:  CC73 65 28            ADC  $28
345:  CC75 85 28            STA  $28
346:  CC77 85 4B            STA  $4B
347:  CC79 A5 29            LDA  $29
348:  CC7B 69 00            ADC  #0
349:  CC7D 85 29   SET2A    STA  $29
350:  CC7F 85 4C            STA  $4C
351:  CC81 8A      SET3     TXA
352:  CC82 85 26            STA  $26
353:  CC84 84 27            STY  $27
354:  CC86 18               CLC
355:  CC87 24 6F            BIT  NUMFL     ;NUMER.ARRAY
356:  CC89 70 0A            BVS  W1        ;NEIN
357:  CC8B 20 46 CF         JSR  ZAHL      ;JA
358:  CC8E B0 CB            BCS  SET
                      ;---------------------------------------
360:  CC90 18      WORK     CLC            ;SORTIEREN BIS ZUM
361:  CC91 A5 71            LDA  $71       ;ERSTEN TAUSCH
362:  CC93 A4 72            LDY  $72       ;B-ELEMENT($71/$72)
363:  CC95 85 22   W1       STA  $22       ;WIRD ZU A-ELEMENT
364:  CC97 65 69            ADC  $69       ;+SCHRITTWEITE
365:  CC99 AA               TAX            ;= B-ELEMENT
366:  CC9A 98               TYA
367:  CC9B 65 6A            ADC  $6A
368:  CC9D C5 FE            CMP  $FE       ;TEST,OB B-ELEMENT
369:  CC9F 90 06            BCC  NEXT      ;>ARRAYENDE,
370:  CCA1 D0 B8            BNE  SET       ;WENN JA,DANN
371:  CCA3 E4 FD            CPX  $FD
372:  CCA5 B0 B4            BCS  SET       ;NAECHSTES TEILFELD
373:  CCA7 85 25   NEXT     STA  $25
374:  CCA9 84 23            STY  $23
375:  CCAB 86 24            STX  $24
376:  CCAD 85 72            STA  $72
377:  CCAF 86 71            STX  $71
378:  CCB1 A4 14            LDY  $14
379:  CCB3 F0 16            BEQ  VERGL
380:  CCB5 A5 4B            LDA  $4B
381:  CCB7 85 61            STA  $61
382:  CCB9 65 6B            ADC  $6B
383:  CCBB 85 63            STA  $63
384:  CCBD 85 4B            STA  $4B
385:  CCBF A5 4C            LDA  $4C
386:  CCC1 85 62            STA  $62
387:  CCC3 65 6C            ADC  $6C
388:  CCC5 85 64            STA  $64
389:  CCC7 85 4C            STA  $4C
                      ;---------------------------------------
391:  CCC9 A0 00            LDY  #0
392:  CCCB B1 24   VERGL    LDA  ($24),Y ;DESKRIPTOR
393:  CCCD F0 C1   B0       BEQ  WORK      ;B-ELEMENT NACH
394:  CCCF 85 5E            STA  $5E       ;$5E-$60;WENN
395:  CCD1 C8               INY            ;LAENGE=0,DANN
396:  CCD2 B1 24            LDA  ($24),Y ;NAECHSTES ELEMENT
397:  CCD4 85 5F            STA  $5F
398:  CCD6 C8               INY
399:  CCD7 B1 24            LDA  ($24),Y
400:  CCD9 85 60            STA  $60
                      ;--- EINSPRUNG EINSORTIEREN (LDY #2) ---
402:  CCDB B1 22   VERS     LDA  ($22),Y ;DESKRIPTOR
403:  CCDD 85 5D            STA  $5D       ;A-ELEMENT NACH
404:  CCDF 88               DEY            ;$5B-$5D;WENN
405:  CCE0 B1 22            LDA  ($22),Y ;LAENGE=0,DANN
406:  CCE2 85 5C            STA  $5C       ;TAUSCHEN
407:  CCE4 88               DEY            ;Y-REG. = 0
408:  CCE5 B1 22            LDA  ($22),Y
409:  CCE7 85 5B            STA  $5B
410:  CCE9 F0 66            BEQ  SWAP
411:  CCEB C5 5E            CMP  $5E       ;VERGLEICHSLAENGE
412:  CCED F0 04            BEQ  VER1      ;NACH X-REG.
413:  CCEF 90 02            BCC  VER1      ;HOLEN
414:  CCF1 A5 5E            LDA  $5E
415:  CCF3 AA      VER1     TAX
                      ;---------------------------------------
417:  CCF4 B1 5C   LOOP     LDA  ($5C),Y ;STRINGVERGLEICH
418:  CCF6 D1 5F            CMP  ($5F),Y ;FUER 1.DIMENSION
419:  CCF8 90 97   B1       BCC  WORK+1
420:  CCFA D0 55            BNE  SWAP
421:  CCFC C8               INY
422:  CCFD CA               DEX
423:  CCFE D0 F4            BNE  LOOP
424:  CD00 A4 5B            LDY  $5B
425:  CD02 C4 5E            CPY  $5E
426:  CD04 90 8B            BCC  WORK+1    ;WENN GLEICH,DANN
427:  CD06 D0 49            BNE  SWAP
428:  CD08 AE 3E CF         LDX  FLAG1     ;TEST AUF 2.DIM.
429:  CD0B F0 C0            BEQ  B0        ;NEIN,NACH "WORK"
                      ;---------------------------------------
431:  CD0D 86 2A            STX  $2A       ;DESKRIPTOREN DER
432:  CD0F 20 15 CF         JSR  D2SET     ;2.DIMENSION
433:  CD12 C6 2A   ZL       DEC  $2A       ;SETZEN UND
434:  CD14 F0 B7   B2       BEQ  B0        ;VERGLEICH NACH
435:  CD16 20 F6 CE         JSR  DSET      ;DEN ELEMENTEN
436:  CD19 A0 02            LDY  #2        ;ROUTINE IST
437:  CD1B B1 47   Z1       LDA  ($47),Y ;SELBSTMODI-
438:  CD1D 99 3B CD         STA  V1,Y      ;FIZIEREND
439:  CD20 B1 49            LDA  ($49),Y
440:  CD22 99 3E CD         STA  V2,Y
441:  [illegible] 88        DEY
442:  CD26 D0 F3            BNE  Z1
443:  CD28 B1 49            LDA  ($49),Y
444:  CD2A F0 A1   B4       BEQ  B0
445:  CD2C 85 46            STA  $46
446:  CD2E B1 47            LDA  ($47),Y
447:  CD30 F0 1F            BEQ  SWAP
448:  CD32 85 45            STA  $45
449:  CD34 C5 46            CMP  $46
450:  CD36 90 02            BCC  V0
451:  CD38 A5 46            LDA  $46
452:  CD3A AA      V0       TAX
453:  CD3B B9 11 47 V1      LDA  $4711,Y ;DUMMY-WERT
454:  CD3E D9 15 08 V2      CMP  $0815,Y ;    "
455:  CD41 90 B5            BCC  B1
456:  CD43 D0 0C            BNE  SWAP
457:  CD45 C8               INY
458:  CD46 CA               DEX
459:  CD47 D0 F2            BNE  V1
460:  CD49 A4 45            LDY  $45
461:  CD4B C4 46            CPY  $46
462:  CD4D 90 A9            BCC  B1
463:  CD4F F0 C1            BEQ  ZL
                      ;---------------------------------------
465:  CD51 AC 3E CF SWAP    LDY  FLAG1     ;TEST 2-DIMENSIONAL
466:  CD54 F0 03            BEQ  SWAP1     ;NEIN
467:  CD56 20 D9 CE         JSR  SWAPD2    ;JA,TAUSCHEN
468:  CD59 A4 14   SWAP1    LDY  $14       ;TEST NEBENARRAY
469:  CD5B F0 20            BEQ  SWAP3     ;NEIN
470:  CD5D 88               DEY            ;JA,TAUSCHEN
471:  CD5E B1 61   SWAP2    LDA  ($61),Y
472:  CD60 AA               TAX
473:  CD61 B1 63            LDA  ($63),Y
474:  CD63 91 61            STA  ($61),Y
475:  CD65 8A               TXA
476:  CD66 91 63            STA  ($63),Y
477:  CD68 88               DEY
478:  CD69 10 F3            BPL  SWAP2
479:  CD6B C8               INY            ;Y-REG.=0
480:  CD6C 38               SEC
481:  CD6D A5 61            LDA  $61       ;NEBENARRAY
482:  CD6F 85 63            STA  $63       ;ZUR EINSOR-
483:  CD71 E5 6B            SBC  $6B       ;TIERUNG
484:  CD73 85 61            STA  $61       ;VORBEREITEN
485:  CD75 A5 62            LDA  $62       ;SIEHE PRG.-
486:  CD77 85 64            STA  $64       ;TEIL SORT
487:  CD79 E5 6C            SBC  $6C
488:  CD7B 85 62            STA  $62
489:  CD7D A5 5B   SWAP3    LDA  $5B       ;HAUPTARRAY
490:  CD7F 91 24            STA  ($24),Y ;DESKRIPTOREN
491:  CD81 A5 5E            LDA  $5E       ;TAUSCHEN
492:  CD83 91 22            STA  ($22),Y ;
493:  CD85 C8               INY
494:  CD86 A5 5C            LDA  $5C
495:  CD88 91 24            STA  ($24),Y
496:  CD8A A5 5F            LDA  $5F
```

```
497:   CD8C 91 22          STA  ($22),Y
498:   CD8E C8             INY            ;Y-REG.=2
499:   CD8F A5 5D          LDA  $5D
5ØØ:   CD91 91 24          STA  ($24),Y
5Ø1:   CD93 A5 6Ø          LDA  $6Ø
5Ø2:   CD95 91 22          STA  ($22),Y
                    ;---------------------------------------
5Ø4:   CD97 A5 22   SORT   LDA  $22       ;NACH LINKS EINSOR-
5Ø5:   CD99 A6 23          LDX  $23       ;TIEREN,BIS KEIN
5Ø6:   CD9B E4 27          CPX  $27       ;TAUSCH ERFOLGT BZW
5Ø7:   CD9D DØ Ø4          BNE  SO1       ;DIE LINKE TEILFELD
5Ø8:   CD9F C5 26          CMP  $26       ;GRENZE ERREICHT
5Ø9:   CDA1 FØ 87          BEQ  B4        ;IST.DESKRIPTOR DES
51Ø:   CDA3 85 24   SO1    STA  $24       ;EINZUSORTIERENDEN
511:   CDA5 86 25          STX  $25       ;ELEMENTS BLEIBT IN
512:   CDA7 E5 69          SBC  $69       ;$5E-$6Ø.B-ELEMENT
513:   CDA9 85 22          STA  $22       ;WIRD DURCH ZEIGER
514:   CDAB 8A             TXA            ;$24/$25 GESETZT.
515:   CDAC E5 6A          SBC  $6A       ;ZEIGER $71/$72
516:   CDAE 85 23          STA  $23       ;WIRD NICHT
517:   CDBØ 4C DB CC       JMP  VERS      ;GEAENDERT !
                    ;
                    ;------------ UNTERPROGRAMME -----------
                    ;
521:   CDB3 A9 ØØ   ARRAY  LDA  #Ø        ;DEFAULTWERT FUER
522:   CDB5 8D 3D CF       STA  FLAG      ;GANZES ARRAY
523:   CDB8 8D 3E CF       STA  FLAG1     ;DEFAULT 1.DIM.
524:   CDBB 2Ø 79 ØØ       JSR  CHRGOT    ;LETZTES ZEICHEN
525:   CDBE C9 23          CMP  #"#"      ;NUR TEILARRAY
526:   CDCØ DØ ØA          BNE  ARR1      ;NEIN
527:   CDC2 2Ø 9B B7       JSR  $B79B     ;1 ODER 2 DIM.
528:   CDC5 CA             DEX
529:   CDC6 8E 3E CF       STX  FLAG1     ;FLAGGE 1/2 DIM.
53Ø:   CDC9 CE 3D CF       DEC  FLAG      ;FLAGGE TEILSORT.
531:   CDCC 2Ø FD AE ARR1  JSR  KOMMA
532:   CDCF 2Ø 9E AD ARR2  JSR  TERM      ;NICHT $BØ8B !!!!
533:   CDD2 AC 3D CF       LDY  FLAG
534:   CDD5 DØ 6F          BNE  TEIL1
                    ;-------- GANZES ARRAY SORTIEREN -------
536:   CDD7 A5 2F   ARR3   LDA  $2F       ;AB ANFANG DER
537:   CDD9 A6 3Ø          LDX  $3Ø       ;ARRAYS MIT DER
538:   CDDB 85 57   GANZ   STA  $57       ;SUCHE BEGINNEN
539:   CDDD 86 58          STX  $58
54Ø:   CDDF C5 31          CMP  $31       ;WENN ENDE DER
541:   CDE1 DØ Ø4          BNE  GANZØ     ;ARRAYS ERREICHT,
542:   CDE3 E4 32          CPX  $32       ;DANN NICHT
543:   CDE5 FØ 33          BEQ  FEHLER    ;GEFUNDEN,FEHLER !
544:   CDE7 AØ ØØ   GANZØ  LDY  #Ø
545:   CDE9 B1 57          LDA  ($57),Y
546:   CDEB C8             INY
547:   CDEC C5 45          CMP  $45       ;VARIABLENNAME
548:   CDEE DØ Ø4          BNE  GANZ1
549:   CDFØ B1 57          LDA  ($57),Y
55Ø:   CDF2 C5 46          CMP  $46
551:   CDF4 Ø8      GANZ1  PHP            ;WENN GEFUNDEN,DANN
552:   CDF5 C8             INY            ;ZEROFLAG GESETZT
553:   CDF6 B1 57          LDA  ($57),Y   ;ENDE NACH $59/$5A
554:   CDF8 18             CLC
555:   CDF9 65 57          ADC  $57
556:   CDFB 85 59          STA  $59
557:   CDFD C8             INY
558:   CDFE B1 57          LDA  ($57),Y
559:   CEØØ 65 58          ADC  $58
56Ø:   CEØ2 85 5A          STA  $5A
561:   CEØ4 AA             TAX
562:   CEØ5 A5 59          LDA  $59
563:   CEØ7 28             PLP            ;TEST,OB GEFUNDEN
564:   CEØ8 DØ D1          BNE  GANZ      ;NAECHSTES ARRAY
565:   CEØA C8             INY            ;--ARRAY GEFUNDEN
566:   CEØB B1 57          LDA  ($57),Y   ;TEST,OB ARRAY
567:   CEØD C9 Ø1          CMP  #1        ;EINDIMENSIONAL
568:   CEØF FØ 1C          BEQ  GANZ4     ;JA,DANN OK
569:   CE11 C9 Ø2          CMP  #2
57Ø:   CE13 DØ Ø5          BNE  FEHLER
571:   CE15 AD 3E CF       LDA  FLAG1
572:   CE18 FØ Ø5          BEQ  GANZ3
573:   CE1A A2 19   FEHLER LDX  #$19      ;FORMULA TOO
574:   CE1C 6C ØØ Ø3       JMP  ($3ØØ)    ;KOMPLEX AUSGEBEN
575:   CE1F C8      GANZ3  INY
576:   CE2Ø B1 57          LDA  ($57),Y
577:   CE22 DØ F6          BNE  FEHLER
578:   CE24 C8             INY
579:   CE25 B1 57          LDA  ($57),Y
58Ø:   CE27 8D 3E CF       STA  FLAG1
581:   CE2A A9 Ø9          LDA  #9        ;9
582:   CE2C 2C             .BYTE$2C       ;BZW
583:   CE2D A9 Ø7   GANZ4  LDA  #7        ;7 BYTE ZU
584:   CE2F 18             CLC            ;ANFANG ADDIEREN
585:   CE3Ø 65 57          ADC  $57       ;UM AUF 1.DESKR.
586:   CE32 85 57          STA  $57       ;ZU ZEIGEN
587:   CE34 9Ø Ø2          BCC  NAME
588:   CE36 E6 58          INC  $58
                    ;
59Ø:   CE38 A2 Ø5   NAME   LDX  #5        ;SPEZ.PLATZBEDARF
591:   CE3A A5 46          LDA  $46       ;AUS VARIABLENNAME
592:   CE3C 1Ø Ø2          BPL  NAME1     ;BERECHNEN UND INS
593:   CE3E CA             DEX            ;X-REGISTER
594:   CE3F CA             DEX            ;INTEGER = 2
595:   CE4Ø A5 45   NAME1  LDA  $45       ;STRING = 3
596:   CE42 1Ø Ø1          BPL  NAME2     ;REAL   = 5
597:   CE44 CA             DEX
598:   CE45 6Ø      NAME2  RTS
                    ;---------------------------------------
                    ;-- BEREICH FUER TEILSORTIERUNG HOLEN --
6Ø2:   CE46 AØ Ø3   TEIL1  LDY  #3
6Ø3:   CE48 B9 45 ØØ TEIL2 LDA  $45,Y
6Ø4:   CE4B 48             PHA            ;NAME UND ANFANG
6Ø5:   CE4C 88             DEY            ;AUF STACK
6Ø6:   CE4D 1Ø F9          BPL  TEIL2
6Ø7:   CE4F 2Ø FD AE       JSR  KOMMA     ;ENDE HOLEN
6Ø8:   CE52 2Ø 9E AD       JSR  TERM
6Ø9:   CE55 2Ø 38 CE       JSR  NAME      ;ANGEGEBENES
61Ø:   CE58 8A             TXA            ;ELEMENT AUCH NOCH
611:   CE59 18             CLC            ;MITSORTIEREN
612:   CE5A 65 47          ADC  $47
613:   CE5C 85 59          STA  $59
614:   CE5E A4 48          LDY  $48
615:   CE6Ø 9Ø Ø1          BCC  TEIL3
616:   CE62 C8             INY
617:   CE63 84 5A   TEIL3  STY  $5A
618:   CE65 68             PLA            ;TEST AUF
619:   CE66 C5 45          CMP  $45       ;GLEICHEN NAMEN
62Ø:   CE68 DØ BØ          BNE  FEHLER
621:   CE6A 68             PLA
622:   CE6B C5 46          CMP  $46
623:   CE6D DØ AB   FEHLA  BNE  FEHLER
624:   CE6F 68             PLA            ;LOW-BYTE ANFANG
625:   CE7Ø A8             TAY
626:   CE71 68             PLA            ;HIGH-BYTE ANFANG
627:   CE72 C5 3Ø          CMP  $3Ø       ;TEST,OB IM
628:   CE74 9Ø A4          BCC  FEHLER    ;BEREICH DER
629:   CE76 DØ Ø4          BNE  TEIL4     ;ARRAYS
63Ø:   CE78 C4 2F          CPY  $2F
631:   CE7A 9Ø 9E          BCC  FEHLER
632:   CE7C 85 58   TEIL4  STA  $58
633:   CE7E 84 57          STY  $57
634:   CE8Ø C5 5A          CMP  $5A       ;TEST,OB
635:   CE82 9Ø C1          BCC  NAME2     ;ANFANG < ENDE
636:   CE84 DØ 94          BNE  FEHLER
637:   CE86 C4 59          CPY  $59
638:   CE88 BØ 9Ø   FEHLB  BCS  FEHLER
639:   CEBA 6Ø             RTS
                    ;---------------------------------------
641:   CEBB 4A      LTEST  LSR            ;GES.PLATZ
642:   CEBC 29 Ø2          AND  #2        ;HAUPTARRAY
643:   CEBE Ø8             PHP            ;*SPEZ.PLATZ
644:   CEBF A5 22          LDA  $22       ;NEBENARRAY
645:   CE91 ØA             ASL            ;    =
646:   CE92 AA             TAX            ;GES.PLATZ
647:   CE93 A5 23          LDA  $23       ;NEBENARRAY
648:   CE95 2A             ROL            ;*SPEZ.PLATZ
649:   CE96 BØ FØ          BCS  FEHLB     ;HAUPTARRAY
65Ø:   CE98 A8             TAY            ;SONST FEHLER !
651:   CE99 28             PLP            ;--------------
652:   CE9A 9Ø 13          BCC  LT2       ;INTEGER
653:   CE9C FØ Ø8          BEQ  LT1       ;STRING
654:   CE9E 8A             TXA            ;REAL
655:   CE9F ØA             ASL
656:   CEAØ AA             TAX
657:   CEA1 98             TYA
658:   CEA2 2A             ROL
659:   CEA3 A8             TAY
66Ø:   CEA4 BØ E2   FEHLC  BCS  FEHLB
661:   CEA6 18      LT1    CLC
662:   CEA7 8A             TXA
663:   CEA8 65 22          ADC  $22
664:   CEAA AA             TAX
665:   CEAB 98             TYA
666:   CEAC 65 23          ADC  $23
667:   CEAE A8             TAY
668:   CEAF 6Ø      LT2    RTS
                    ;
67Ø:   CEBØ 2Ø 8B CE TEST  JSR  LTEST     ;EINSPRUNG
671:   CEB3 86 24          STX  $24       ;TEST AUF GLEICHE
672:   CEB5 84 25          STY  $25       ;ELEMENTZAHL
673:   CEB7 A5 69          LDA  $69       ;S."LTEST"
674:   CEB9 85 22          STA  $22
675:   CEBB A5 6A          LDA  $6A
676:   CEBD 85 23          STA  $23
677:   CEBF A5 14          LDA  $14
678:   CEC1 2Ø 8B CE       JSR  LTEST
679:   CEC4 C4 25          CPY  $25
68Ø:   CEC6 DØ A5          BNE  FEHLA
681:   CEC8 E4 24          CPX  $24
682:   CECA DØ A1   FEHLD  BNE  FEHLA
683:   CECC A5 FB          LDA  $FB       ;TEST OB BEIDE
684:   CECE C5 57          CMP  $57       ;ARRAYS GLEICH SIND
685:   CEDØ DØ Ø6          BNE  OK1       ;=SYS52ØØØ,A$,A$
686:   CED2 A5 FC          LDA  $FC       ;WENN JA,FEHLER
687:   CED4 C5 58          CMP  $58
688:   CED6 FØ BØ          BEQ  FEHLB
689:   CED8 6Ø      OK1    RTS
                    ;---------------------------------------
691:   CED9 88      SWAPD2 DEY            ;ALLE ELEMENTE
692:   CEDA 84 2A          STY  $2A       ;DER 2.DIM.
693:   CEDC 2Ø 15 CF       JSR  D2SET     ;TAUSCHEN
694:   CEDF 2Ø F6 CE D1    JSR  DSET
695:   CEE2 AØ Ø2          LDY  #2
696:   CEE4 B1 47   D2     LDA  ($47),Y
697:   CEE6 AA             TAX
698:   CEE7 B1 49          LDA  ($49),Y
699:   CEE9 91 47          STA  ($47),Y
7ØØ:   CEEB 8A             TXA
7Ø1:   CEEC 91 49          STA  ($49),Y
7Ø2:   CEEE 88      D4     DEY            ;EINSPRUNG NUM.
7Ø3:   CEEF 1Ø F3          BPL  D2        ;NEBENARRAY
7Ø4:   CEF1 C6 2A          DEC  $2A       ;TAUSCHEN
7Ø5:   CEF3 DØ EA          BNE  D1
7Ø6:   CEF5 6Ø             RTS
                    ;---------------------------------------
7Ø8:   CEF6 A5 47   DSET   LDA  $47       ;DESKRIPTOREN
7Ø9:   CEF8 18             CLC            ;FUER ELEMENTE
71Ø:   CEF9 6D 3F CF       ADC  OL        ;DER 2.DIM.
711:   CEFC 85 47          STA  $47       ;BERECHNEN
```

Listing 2. Der Source-Code zu Shellsort in Assembler (Fortsetzung)

```
712:   CEFE A5 48                LDA  $48
713:   CF00 6D 40 CF             ADC  OH
714:   CF03 85 48                STA  $48
715:   CF05 18                   CLC
716:   CF06 A5 49                LDA  $49
717:   CF08 6D 3F CF             ADC  OL
718:   CF0B 85 49                STA  $49
719:   CF0D A5 4A                LDA  $4A
720:   CF0F 6D 40 CF             ADC  OH
721:   CF12 85 4A                STA  $4A
722:   CF14 60                   RTS
                        ;
724:   CF15 A0 03       D2SET    LDY  #3       ;VORBEREITUNG
725:   CF17 B9 22 00    D3       LDA  $22,Y    ;AUF 1.ELEMENT
726:   CF1A 99 47 00             STA  $47,Y    ;DER 2.DIMENSION
727:   CF1D 88                   DEY
728:   CF1E 10 F7                BPL  D3
729:   CF20 60                   RTS
                        ;-----------------------------------------
731:   CF21 A5 FD       GPLATZ   LDA  $FD      ;GESAMTPLATZ-
732:   CF23 38                   SEC           ;BEDARF FUER
733:   CF24 E5 FB                SBC  $FB      ;HAUPTARRAY
734:   CF26 85 69                STA  $69      ;BERECHNEN
735:   CF28 A5 FE                LDA  $FE
736:   CF2A E5 FC                SBC  $FC
737:   CF2C 85 6A                STA  $6A
738:   CF2E A5 59       PLATZ    LDA  $59      ;WIE OBEN FUER
739:   CF30 38                   SEC           ;NEBENARRAY
740:   CF31 E5 57                SBC  $57
741:   CF33 85 22                STA  $22
742:   CF35 AA                   TAX
743:   CF36 A5 5A                LDA  $5A
744:   CF38 E5 58                SBC  $58
745:   CF3A 85 23                STA  $23
746:   CF3C 60          BLE      RTS
                        ;-----------------------------------------
748:   CF3D EA          FLAG     NOP           ;FLAGGE TEILARRAY
749:   CF3E EA          FLAG1    NOP           ;FLAGGE 2 DIM.
750:   CF3F EA          OL       NOP           ;OFFSET AUF 1.
751:   CF40 EA          OH       NOP           ;ELEMENT 2.DIM.
                        ;-----------------------------------------
                        ;-------- NUMER.ARRAY SORTIEREN ---------
                        ;-----------------------------------------
755:   CF41 A5 71       BL       LDA  $71      ;VERGL ROUTINEN
756:   CF43 A4 72                LDY  $72      ;"WORK" BIS "SORT"
757:   CF45 18                   CLC
758:   CF46 85 5B       ZAHL     STA  $5B      ;A-ELEMENT
759:   CF48 84 5C                STY  $5C
760:   CF4A 65 69                ADC  $69
761:   CF4C AA                   TAX
762:   CF4D 98                   TYA
763:   CF4E 65 6A                ADC  $6A
764:   CF50 C5 FE                CMP  $FE
765:   CF52 90 06                BCC  BL1
766:   CF54 D0 E6                BNE  BLE
767:   CF56 E4 FD                CPX  $FD
768:   CF58 B0 E2                BCS  BLE
769:   CF5A A8          BL1      TAY
770:   CF5B 85 72                STA  $72      ;B-ELEMENT
771:   CF5D 86 71                STX  $71      ;GROSSE SCHLEIFE
772:   CF5F 85 5E                STA  $5E      ;B-ELEMENT
773:   CF61 86 5D                STX  $5D      ;EINSORTIEREN
774:   CF63 A5 14                LDA  $14
775:   CF65 F0 14                BEQ  ZVER
776:   CF67 A5 4B                LDA  $4B
777:   CF69 85 47                STA  $47
778:   CF6B 65 6B                ADC  $6B
779:   CF6D 85 4B                STA  $4B
780:   CF6F 85 49                STA  $49
781:   CF71 A5 4C                LDA  $4C
782:   CF73 85 48                STA  $48
783:   CF75 65 6C                ADC  $6C
784:   CF77 85 4C                STA  $4C
785:   CF79 85 4A                STA  $4A
                        ;
787:   CF7B 8A          ZVER     TXA
788:   CF7C 24 6E                BIT  RUND     ;WENN RUND=$FF
789:   CF7E 70 60                BVS  INT      ;DANN INTEGER !
790:   CF80 20 A2 BB             JSR  $BBA2    ;B-ELEM.IN FAC
791:   CF83 A5 5B                LDA  $5B      ;VERGL.MIT
792:   CF85 A4 5C                LDY  $5C      ;A-ELEMENT
793:   CF87 20 5B BC    ZV1      JSR  $BC5B
794:   CF8A AA                   TAX
795:   CF8B F0 B4                BEQ  BL
796:   CF8D CD FE CF             CMP  FLAG2
797:   CF90 F0 AF       ZVB1     BEQ  BL
798:   CF92 A4 6D       ZTAU     LDY  TYPE
799:   CF94 88                   DEY
800:   CF95 B1 5B       SZ1      LDA  ($5B),Y
801:   CF97 AA                   TAX
802:   CF98 B1 5D                LDA  ($5D),Y
803:   CF9A 91 5B                STA  ($5B),Y
804:   CF9C 8A                   TXA
805:   CF9D 91 5D                STA  ($5D),Y
806:   CF9F 88                   DEY
807:   CFA0 10 F3                BPL  SZ1
808:   CFA2 A4 14                LDY  $14
809:   CFA4 F0 05                BEQ  ZV2
810:   CFA6 E6 2A                INC  $2A
811:   CFA8 20 EE CE             JSR  D4
                        ;-----------------------------------------
813:   CFAB A5 5B       ZV2      LDA  $5B      ;ENTSPR."SORT"
814:   CFAD A4 5C                LDY  $5C      ;B-ELEMENT BLEIBT
815:   CFAF C4 27                CPY  $27      ;IM FAC
816:   CFB1 D0 04                BNE  ZV3
817:   CFB3 C5 26                CMP  $26
818:   CFB5 F0 8A                BEQ  BL
819:   CFB7 85 5D       ZV3      STA  $5D
820:   CFB9 84 5E                STY  $5E
821:   CFBB E5 69                SBC  $69
822:   CFBD AA                   TAX
823:   CFBE 98                   TYA
824:   CFBF E5 6A                SBC  $6A
825:   CFC1 A8                   TAY
826:   CFC2 86 5B       ZV3A     STX  $5B
827:   CFC4 84 5C                STY  $5C
828:   CFC6 A5 14                LDA  $14
829:   CFC8 F0 11                BEQ  ZV4
830:   CFCA A5 47                LDA  $47
831:   CFCC 38                   SEC
832:   CFCD 85 49                STA  $49
833:   CFCF E5 6B                SBC  $6B
834:   CFD1 85 47                STA  $47
835:   CFD3 A5 48                LDA  $48
836:   CFD5 85 4A                STA  $4A
837:   CFD7 E5 6C                SBC  $6C
838:   CFD9 85 48                STA  $48
839:   CFDB 8A          ZV4      TXA
840:   CFDC 24 6E                BIT  RUND
841:   CFDE 50 A7                BVC  ZV1
                        ;
843:   CFE0 AE FE CF    INT      LDX  FLAG2
844:   CFE3 A0 00                LDY  #0
845:   CFE5 B1 5B       I1       LDA  ($5B),Y  ;VERGLEICH
846:   CFE7 D1 5D                CMP  ($5D),Y  ;INTEGER
847:   CFE9 90 0D                BCC  KL       ;KLEINER
848:   CFEB D0 07                BNE  GR       ;GROESSER
849:   CFED C8                   INY
850:   CFEE C0 02                CPY  #2
851:   CFF0 D0 F3                BNE  I1
852:   CFF2 F0 9C                BEQ  ZVB1
853:   CFF4 8A          GR       TXA           ;ERGEBNIS
854:   CFF5 49 FF                EOR  #$FF     ;HERUMDREHEN
855:   CFF7 AA                   TAX           ;WIEDER NACH X-REG.
856:   CFF8 8A          KL       TXA
857:   CFF9 30 97                BMI  ZTAU
858:   CFFB 4C 41 CF             JMP  BL
                        ;
860:   CFFE 01          FLAG2    .BYTE1        ; 1 =KLEINSTES
                        ;                      255=GROESSTES
                        ;                      ELEMENT AN
                        ;                      ARRAYANFANG
```

Listing 2. Der Source-Code zu Shellsort in Assembler (Schluß)

```
Name : quicksort              c000 c317
---------------------------------------
c000 : 20 fd ae 20 9e b7 86 a5   db
c008 : 20 fd ae 20 9e b7 86 a6   e5
c010 : 20 a6 c2 a2 02 b5 bb 95   70
c018 : b3 ca 10 f9 a5 b4 a6 b3   76
c020 : e0 7f f0 01 4a 85 b6 ad   23
c028 : 00 c6 ae 01 c6 8d 00 c4   b9
c030 : 8e 01 c4 ad 00 c7 ae 01   21
c038 : c7 8d 00 c5 8e 01 c5 a9   da
c040 : 00 85 bf 20 79 00 c9 2c   0e
c048 : d0 05 20 a6 c2 e6 bf a9   2d
c050 : fe 85 be e6 be e6 be a6   09
c058 : be bd 01 c4 dd 01 c5 90   ec
c060 : 0d d0 08 bd 00 c4 dd 00   2d
c068 : c5 90 06 4c 94 c1 bd 00   cf
c070 : c4 85 af bd 01 c4 85 b0   48
c078 : bd 00 c5 85 b1 bd 01 c5   f0
c080 : 85 b2 a5 bf f0 14 bd 00   66
c088 : c6 85 b7 bd 01 c6 85 b8   84
c090 : bd 00 c7 85 b9 bd 01 c7   0d
c098 : 85 ba a5 b1 18 65 af 85   90
c0a0 : aa a5 b0 65 b2 4a 85 ab   e0
c0a8 : 66 aa b0 0d a5 b3 c9 7f   4f
c0b0 : d0 12 a5 aa 45 af 4a 90   64
c0b8 : 0b a5 aa 38 e5 b6 85 aa   c7
c0c0 : b0 02 c6 ab a4 b4 b1 aa   a4
c0c8 : 99 61 00 88 10 f8 a5 b3   ea
c0d0 : 30 10 d0 1e 20 d9 c2 a6   29
c0d8 : b4 b5 61 95 69 ca 10 f9   93
c0e0 : 30 10 a9 61 a0 00 20 8c   52
c0e8 : ba a6 b4 b5 69 95 61 ca   37
c0f0 : 10 f9 20 cf c1 b0 33 20   ad
c0f8 : 03 c2 20 cf c1 b0 2b a4   f6
c100 : b4 b1 af aa b1 b1 91 af   1c
c108 : 8a 91 b1 88 10 f3 a5 bf   8f
c110 : f0 0f a4 bc b1 b7 aa b1   2f
c118 : b9 91 b7 8a 91 b9 88 10   02
c120 : f3 20 99 c1 20 b4 c1 4c   09
c128 : f2 c0 a6 be a5 b1 9d 02   5e
c130 : c5 a5 b2 9d 03 c5 bd 00   7d
c138 : c4 9d 02 c4 bd 01 c4 9d   16
c140 : 03 c4 a5 bf f0 16 a5 b9   d0
c148 : 9d 02 c7 a5 ba 9d 03 c7   c1
c150 : bd 00 c6 9d 02 c6 bd 01   c2
c158 : c6 9d 03 c6 20 53 c0 a6   73
c160 : be a5 af 9d 02 c4 a5 b0   cf
c168 : 9d 03 c4 bd 00 c5 9d 02   18
c170 : c5 bd 01 c5 9d 03 c5 a5   61
c178 : bf f0 16 a5 b7 9d 02 c6   e8
c180 : a5 b8 9d 03 c6 bd 00 c7   33
c188 : 9d 02 c7 bd 01 c7 9d 03   9b
c190 : c7 20 53 c0 c6 be c6 be   4f
c198 : 60 a5 af 18 65 b5 85 af   33
c1a0 : 90 02 e6 b0 a5 bf f0 0b   33
c1a8 : a5 b7 18 65 bd 85 b7 90   e4
c1b0 : 02 e6 b8 60 a5 b1 38 e5   f4
c1b8 : b5 85 b1 b0 02 c6 b2 a5   1f
c1c0 : bf f0 0b a5 b9 38 e5 bd   df
c1c8 : 85 b9 b0 02 c6 ba 60 a5   a5
c1d0 : b0 c5 b2 90 0a d0 08 a5   b4
c1d8 : af c5 b1 90 02 f0 01 60   55
```

```
c1e0 : 18 60 a5 af a4 b0 20 5b  8f
c1e8 : bc f0 08 30 06 20 99 c1  70
c1f0 : 4c e2 c1 a5 b1 a4 b2 20  1e
c1f8 : 5b bc 10 06 20 b4 c1 4c  bd
c200 : f3 c1 60 a5 b3 f0 4e 30  fd
c208 : d9 a0 00 b1 af 20 3c c2  da
c210 : 90 09 d0 0d c8 b1 af c5  5f
c218 : 62 b0 06 20 99 c1 4c 09  43
c220 : c2 a0 00 b1 b1 20 3c c2  fb
c228 : 90 11 d0 09 c8 b1 b1 c5  02
c230 : 62 90 08 f0 06 20 b4 c1  b2
c238 : 4c 21 c2 60 a6 61 e0 80  cb
c240 : b0 07 c9 80 b0 09 c5 61  23
c248 : 60 c9 80 90 06 b0 f7 a9  d8
c250 : 01 18 60 38 60 a5 af a6  bc
c258 : b0 85 ac 86 ad 20 7e c2  22
c260 : b0 06 20 99 c1 4c 55 c2  a8
c268 : a5 b1 a6 b2 85 ac 86 ad  19
c270 : 20 7e c2 90 08 f0 06 20  f2
c278 : b4 c1 4c 68 c2 60 a0 02  e3
c280 : b1 ac 99 61 00 88 10 f8  90
c288 : 20 d9 c2 a0 ff a6 61 e4  de
c290 : 69 90 02 a6 69 e8 ca d0  41
c298 : 05 a5 61 c5 69 60 c8 b1  a1
c2a0 : 62 d1 6a f0 f1 60 20 fd  42
c2a8 : ae 20 8b b0 8d 00 c6 8c  6c
c2b0 : 01 c6 20 fd ae 20 8b b0  58
c2b8 : 8d 00 c7 8c 01 c7 a2 00  a2
c2c0 : a0 02 a5 0d d0 0b a2 80  5d
c2c8 : a0 04 a5 0e f0 03 ca a0  29
c2d0 : 01 86 bb 84 bc c8 84 bd  33
c2d8 : 60 a5 a6 f0 34 a6 a6 a0  27
c2e0 : ff ca f0 0d c8 c4 61 b0  bc
c2e8 : 29 b1 62 c5 a5 d0 f5 f0  d6
c2f0 : f0 84 a7 e6 a7 c8 c4 61  80
c2f8 : b0 06 b1 62 c5 a5 d0 f5  1d
c300 : 98 38 e5 a7 85 61 a5 a7  6c
c308 : 18 65 62 85 62 90 02 e6  9d
c310 : 63 60 a9 00 85 61 60 8d  0e
```

Listing 3. Quicksort in Maschinensprache. Zum Eintippen verwenden Sie bitte den MSE (Hinweise auf Seite 92).

```
0100            ;*** QUICKSORT ***
0110            ;*              *
0120            ;*      1985 BY *
0130            ;*  SAID BALOUI *
0140            ;*              *
0150            ;****************
0160            ;
0170            ;
0180            ;SORTIERUNG BELIEBIGER ARRAYTYPEN
0190            ;MITSORTIERUNG EINES BELIEBIGEN
0200            ;ARRAYS
0210            ;SORTIERUNG KANN AUF EINEN BEL.
0220            ;TEIL DES ARRAYS BESCHRAENKT
0230            ;WERDEN
0240            ;STRINGARRAYS KOENNEN NACH EINEM
0250            ;BEL. STRINGTEIL SORTIERT WERDEN;
0260            ;DIE TEILE MUESSEN HIERZU MIT
0270            ;EINEM BEL. ZEICHEN VONEINANDER
0280            ;GETRENNT WERDEN
0290            ;
0300            ;
0310            ;AUFRUF:
0320            ;-------
0330            ;SYS X,ASCII-CODE TRENNZEICHEN,
0340            ;      FELDNUMMER, •
0350            ;      SORTARRAY(X),
0360            ;      SORTARRAY(Y),
0370            ;      (MITSORTARRAY(X)) •
0380            ;      (MITSORTARRAY(Y))
0390            ;
0400            ;SYS X,94,2,A$(0),A$(52),P%(0),
0410            ;                         P%(52)
0420            ;
0430            ;
0440            ;TRENNZEICHEN: TRENNUNG MEHRERER
0450            ;              FELDER IN EINEM
0460            ;              STRING MIT BEL.ZEI
0470            ;
0480            ;FELDNUMMER: SORTIEREN AB ANGEG.
0490            ;            STRINGTEILFELD
0500            ;            0=GANZEN STR.SORT.
0510            ;
0520            ;SORTARRAY(X): ZU SORTIERENDES
0530            ;              ARRAY (SORTIERUNG
0540            ;              AB ELEMENT X);
0550            ;              BELIEBIGER ARRAY-
0560            ;              TYP KANN SORTIERT
0570            ;              WERDEN
0580            ;
0590            ;SORTARRAY(Y): ZU SORTIERENDES
0600            ;              ARRAY (SORTIERUNG
0610            ;              BIS ELEMENT Y)
0620            ;
0630            ;MITSORTARRAY(X): BELIEBIGES
0640            ;                 MITZUSORTIER.
0650            ;                 ARRAY
0660            ;
0670            ;MITSORTARRAY(Y): BELIEBIGES
0680            ;                 MITZUSORTIER.
0690            ;                 ARRAY
0700            ;
0710            ;
0720            ;IM 'MITSORTARRAY' MUESSEN
0730            ;IDENTISCHE X/Y-WERTE ANGEGEBEN
0740            ;WERDEN WIE IM SORTARRAY!
0750            ;
0760            ;
0770            ;
0780CHRGOT      .DE $79
0790CHKKOM      .DE $AEFD
0800GETBYT      .DE $B79E
0810GETPOS      .DE $B08B
0820KONINARG    .DE $BABC      ;KONSTANTE=>ARG
0830FACKONST    .DE $BC5B      ;VERGL.:FAC/KO.
0840            ;
0850STRTYP      .DE $0D
0860NUMTYP      .DE $0E
0870FAC         .DE $61
0880ARG         .DE $69        ;FLIESSKOMMAAKKU #2
0890DESCR1      .DE FAC        ;DESCRIPTOREN VON STR1
0900DESCR2      .DE ARG        ;DESCRIPTOREN VON STR2
0910            ;
0920            ;
0930TRENN       .DE $A5        ;TRENNZEICHEN
0940FELD        .DE TRENN+1    ;SORTSTRINGTEIL
0950HELP        .DE FELD+1
0960            ;
0970ARRAY       .DE HELP+1     ;ZEIGER AUF ANZAHL DER ARRAYELEMENTE
0980VG          .DE ARRAY+2    ;POINTER AUF VERGLEICHSSTRING (=2.VERGLEICH
SELEM
0990STR         .DE VG+2       ;POINTER AUF 1.VERGLEICHSSTRING
1000X           .DE STR+3
1010Y           .DE X+2
1020TYP         .DE Y+2        ;0=STRING/127=INT/128=REAL
1030ZAHL        .DE TYP+1      ;SCHLEIFENZAEHLER FUER VERSCH.ARRAYTYPEN
1040OFFSET      .DE ZAHL+1     ;OFFSET FUER VERSCH.ARRAYTYPEN FUER POINTER
SETZ
1050KORR        .DE OFFSET+1   ;KORREKTURFAKTOR FUER VERSCH.ARRAYTYPEN FUE
R VG-
1060            ;
1070XKOPIE      .DE KORR+1
1080YKOPIE      .DE XKOPIE+2
1090TYP2        .DE YKOPIE+2
1100ZAHL2       .DE TYP2+1
1110OFFSET2     .DE ZAHL2+1
1120            ;
1130Z           .DE OFFSET2+1 ;EBENENZAEHLER
1140MITFLAG     .DE Z+1        ;FLAG FUER MITZUSORTIERENDES FELD
1150            ;
1160LG          .DE $C400      ;'STACK' FUER LINKE GRENZEN
1170RG          .DE $C500      ;'STACK' FUER RECHTE GRENZEN
1180LG2         .DE $C600      ;'STACK' FUER LINKE GRENZE VON MITSORTARRAY
1190RG2         .DE $C700      ;'STACK' FUER RECHTE GRENZE VON MITSORTARRA
Y
1200            ;
1210            ;
1220            .BA $C000
1230            .OS
1240            ;
1250            ;
1260            ;********************************
1270            ;********INITIALISIERUNG********
1280            ;********************************
1290            ;
1300            ;
1310            ;***INIT FUER SORTARRAY***
1320            JSR CHKKOM           ;TRENNZEICHEN
1330            JSR GETBYT           ;U.NR.DES TEIL-
1340            STX TRENN            ;FELDES (SORT.
1350            JSR CHKKOM           ;VON STRINGS)
1360            JSR GETBYT           ;HOLEN
1370            STX FELD
1380            ;
1390            JSR HOLVAR
1400            LDX #2
1410HOL1        LDA TYP2,X
1420            STA TYP,X
1430            DEX
1440            BPL HOL1
1450            ;
1460            LDA ZAHL             ;KORR=
1470            LDX TYP              ;STRING: 1
1480            CPX #127             ;INTEGER: 1
1490            BEQ NOLSR            ;REAL: 2
1500            LSR A
1510NOLSR       STA KORR
1520            ;
1530            LDA LG2
1540            LDX LG2+1
1550            STA LG               ;GRENZEN
1560            STX LG+1             ;INITIALISIEREN
1570            LDA RG2
1580            LDX RG2+1
1590            STA RG
1600            STX RG+1
1610            ;
1620            ;
1630            ;***INIT FUER MITZUS.ARRAY***
1640            LDA #0               ;FLAG FUER MITZU-
1650            STA MITFLAG          ;SORT.ARRAY INIT.
1660            JSR CHRGOT           ;WENN KOMMA FOLGT:
1670            CMP #','             ;PARAMETER DES
1680            BNE INITEND          ;MITZUSORTIERENDEN
1690            JSR HOLVAR           ;ARRAYS HOLEN UND
1700            INC MITFLAG          ;MITFLAG=1 SETZEN
1710            ;
1720            ;
1730            ;***EBENZAEHLER INITIALISIEREN***
1740INITEND     LDA #254             ;EBENENZAEHLER
1750            STA Z                ;INITIALISIEREN
1760            ;********************************
1770            ;********************************
1780            ;
1790            ;
1800            ;
1810            ;********************************
1820            ;*******QUICKSORT-ROUTINE*******
1830            ;********************************
1840            ;
1850            ;
1860            ;***LG(Z)>=RG(Z)?***
1870EINGANG     INC Z                ;EINE EBENE TIEFER
1880            INC Z
1890            ;
1900            LDX Z
1910            LDA LG+1,X           ;LG(Z) MIT RG(Z)
1920            CMP RG+1,X           ;VERGLEICHEN
1930            BCC EING1
1940            BNE RET
1950            LDA LG,X             ;DURCHGANG BEENDET,
1960            CMP RG,X             ;WENN LG(Z)>=RG(Z)
1970            BCC EING             ;DANN 'RETURN'
1980RET         JMP RETURN
1990            ;******************
2000            ;
2010            ;
```

Listing 4. So sieht der Quicksort-Algorithmus als Assembler-Listing aus

```
2020            ;***X=LG(Z):Y=RG(Z)***
2030EING1       LDA LG,X
2040EING        STA X                ;ZEIGER X UND Y
2050            LDA LG+1,X           ;GLEICH DER LINKEN
2060            STA X+1              ;BZW.RECHTEN GRENZE
2070            LDA RG,X             ;AUF DER MOMENTANEN
2080            STA Y                ;EBENE INITIALIS.
2090            LDA RG+1,X
2100            STA Y+1
2110            ;
2120            LDA MITFLAG
2130            BEQ EINGEND
2140            ;
2150            LDA LG2,X            ;WENN MITFLAG, AUCH
2160            STA XKOPIE           ;FUER MITSORTARRAY
2170            LDA LG2+1,X          ;GRENZEN INIT.
2180            STA XKOPIE+1
2190            LDA RG2.X
2200            STA YKOPIE
2210            LDA RG2+1,X
2220            STA YKOPIE+1
2230            ;********************
2240            ;
2250            ;
2260            ;
2270            ;***ZEIGER AUF VG-ELEM.RICHTEN***
2280            ;
2290            ;**VERGLEICHSELEMENT ERMITTELN**
2300            ;(ENTSPRICHT IN BASIC:)
2310            ;   (VG=INT((X+Y)/2))
2320            ;
2330EINGEND     LDA Y
2340            CLC                  ;DIE ZEIGER X UND Y
2350            ADC X                ;ADDIEREN. ERGEBNIS
2360            STA VG               ;VG(LO) UND
2370            LDA X+1              ;AKKU(HI)
2380            ADC Y+1
2390            ;
2400            LSR A                ;AKKU=HI/2
2410            STA VG+1             ;VG+1=HI/2
2420            ROR VG               ;VG=LO/2
2430            ;
2440            BCS SR               ;REST (NUR BEI REAL
2450            LDA TYP              ;/STRING)? JA =>
2460            CMP #127             ;SORTARRAYTYP INT.
2470            BNE VARHOL           ;NEIN =>
2480            ;
2490            LDA VG               ;ZEIGER AUF VERGL.-
2500            EOR X                ;ELEM.EBENSO GERADE
2510            LSR A                ;ODER UNGERADE WIE
2520            BCC VARHOL           ;DIE LINKE GRENZE?
2530            ;              ;JA =>
2540SR          LDA VG
2550            SEC                  ;ZEIGER AUF VERGL.-
2560            SBC KORR             ;ELEMENT KORRIG.
2570            STA VG
2580            BCS VARHOL
2590            DEC VG+1
2600            ;******************************
2610            ;
2620            ;**INTVAR./STR.DESCR.HOLEN**
2630VARHOL      LDY ZAHL
2640HOLSTR      LDA (VG),Y           ;VG% BZW.DESCR.
2650            STA DESCR1,Y         ;VON VG$ NACH
2660            DEY                  ;DESCR1(+2)
2670            BPL HOLSTR
2680            ;*************************
2690            ;
2700            LDA TYP
2710            BMI REALTYP
2720            BNE NEXT
2730            ;
2740            ;**STRINGDESCR.AUF STRINGTEIL**
2750            JSR STRTEIL          ;DESCR.FUER STRTEIL
2760            ;
2770            LDX ZAHL             ;ENDGUELTIGE
2780HSTR        LDA DESCR1,X         ;STRINGDESCR.
2790            STA DESCR2,X         ;NACH DESCR2 -
2800            DEX                  ;DESCR2+2
2810            BPL HSTR
2820            BMI NEXT             ;ABS.
2830            ;*****************************
2840            ;
2850            ;**REAL=>ARG/KONST=>FAC/FAC=>ARG*
2860REALTYP     LDA #L,DESCR1        ;KONSTANTE
2870            LDY #H,DESCR1        ;NACH ARG
2880            JSR KONINARG
2890            ;
2900            LDX ZAHL             ;ARG NACH FAC
2910ARGINFAC    LDA ARG,X            ;KOPIEREN
2920            STA FAC,X
2930            DEX
2940            BPL ARGINFAC
2950            ;******************************
2960            ;
2970            ;
2980            ;
2990            ;***X>Y? JA=>EINE EBENE TIEFER***
3000NEXT        JSR XYVERGL          ;X MIT Y VERGL.
3010            BCS TEILFELD         ;X>Y? JA=>
3020            ;******************************
3030            ;
3040            ;
3050            ;
3060            ;***VAR(X/Y) MIT VAR(VG) VERGL***
3070            JSR COMPARE
3080            ;******************************
3090            ;
3100            ;
3110            ;
3120            ;***X>Y? JA=>EINE EBENE TIEFER***
3130            JSR XYVERGL          ;X>Y ?
3140            BCS TEILFELD         ;JA =>
3150            ;******************************
3160            ;
3170            ;
3180            ;
3190            ;***SWAP VAR(X) UND VAR(Y)***
3200            LDY ZAHL             ;DIE DESCRIPTOREN
3210SWAP        LDA (X),Y            ;VON A$(X) UND
3220            TAX                  ;UND A$(Y MIT-
3230            LDA (Y),Y            ;EINANDER VERTAU-
3240            STA (X),Y            ;SCHEN
3250            TXA
3260            STA (Y),Y            ;S$=A$(X):A$(X)=
3270            DEY                  ;A$(Y):A$(Y)=S$
3280            BPL SWAP
3290            ;
3300            LDA MITFLAG
3310            BEQ SWAPEND
3320            ;
3330            LDY ZAHL2
3340SWAP2       LDA (XKOPIE),Y
3350            TAX
3360            LDA (YKOPIE),Y       ;WENN MITFLAG,
3370            STA (XKOPIE),Y       ;AUCH DIE DESCR.
3380            TXA                  ;VON MITSORTARRAY
3390            STA (YKOPIE),Y       ;VERTAUSCHEN
3400            DEY
3410            BPL SWAP2
3420            ;***************************
3430            ;
3440            ;
3450            ;
3460            ;***X AUF NEXT,Y AUF LAST ELEM***
3470SWAPEND     JSR XNEXT            ;ZEIGER X AUF
3480            ;                  NEXT ELEMENT
3490            ;
3500            JSR YLAST            ;ZEIGER Y AUF
3510            ;                  LAST ELEMENT
3520            ;
3530            JMP NEXT             ;NAECHSTE RUNDE!
3540            ;******************************
3550            ;
3560            ;
3570            ;
3580            ;***LINKES TEILFELD ERMITTELN***
3590TEILFELD    LDX Z
3600            LDA Y                ;NAECHSTE RECHTE
3610            STA RG+2,X           ;GRENZE=MOMENT.
3620            LDA Y+1              ;RECHTER ZEIGER
3630            STA RG+3.X
3640            LDA LG,X             ;NAECHSTE LINKE
3650            STA LG+2,X           ;GRENZE=MOMENT.
3660            LDA LG+1,X           ;LINKE GRENZE
3670            STA LG+3,X
3680            ;
3690            LDA MITFLAG
3700            BEQ LIEND
3710            ;
3720            LDA YKOPIE           ;WENN MITFLAG,
3730            STA RG2+2,X          ;AUCH NEUE
3740            LDA YKOPIE+1         ;GRENZEN FUER
3750            STA RG2+3,X          ;MITSORTARRAY
3760            LDA LG2,X            ;FESTLEGEN
3770            STA LG2+2,X
3780            LDA LG2+1,X
3790            STA LG2+3,X
3800LIEND       JSR EINGANG
3810            ;******************************
3820            ;
3830            ;
3840            ;***RECHTES TEILFELD ERMITTELN***
3850            LDX Z
3860            LDA X                ;NAECHSTE LINKE
3870            STA LG+2,X           ;GRENZE=MOMENT.
3880            LDA X+1              ;LINKER ZEIGER
3890            STA LG+3,X
3900            LDA RG,X             ;NAECHSTE RECHTE
3910            STA RG+2,X           ;GRENZE=MOMENT.
3920            LDA RG+1,X           ;RECHTE GRENZE
3930            STA RG+3,X
3940            ;
3950            LDA MITFLAG
3960            BEQ REEND
3970            ;
3980            LDA XKOPIE           ;WENN MITFLAG,
3990            STA LG2+2,X          ;GLEICHES FUER
4000            LDA XKOPIE+1         ;MITSORTARRAY
4010            STA LG2+3,X
4020            LDA RG2,X
4030            STA RG2+2,X
4040            LDA RG2+1,X
4050            STA RG2+3,X
4060REEND       JSR EINGANG
4070            ;*******************************
4080            ;
4090            ;
4100            ;
4110            ;***EINE EBENE HOEHER***
4120RETURN      DEC Z
4130            DEC Z
4140            RTS
4150            ;**********************
4160            ;**********************
4170            ;
4180            ;
4190            ;
4200            ;
4210            ;*****************
4220            ;***UNTERROUTINEN***
4230            ;*****************
4240            ;
4250            ;
4260            ;***X AUF NEXT ELEMENT***
4270XNEXT       LDA X
4280            CLC
4290            ADC OFFSET
4300            STA X                ;ZEIGER AUF
4310            BCC XNEX             ;NAECHSTES ELEMENT
4320            INC X+1
4330            ;
4340XNEX        LDA MITFLAG
4350            BEQ XNEXTEND
4360            ;
4370            LDA XKOPIE
4380            CLC
4390            ADC OFFSET2
4400            STA XKOPIE
4410            BCC XNEXTEND
4420            INC XKOPIE+1
4430XNEXTEND    RTS
4440            ;**********************
4450            ;
4460            ;
4470            ;***Y AUF LAST ELEMENT***
4480YLAST       LDA Y
4490            SEC
4500            SBC OFFSET           ;ZEIGER AUF
4510            STA Y                ;VORIGES ELEMENT
4520            BCS YNEX
4530            DEC Y+1
4540            ;
4550YNEX        LDA MITFLAG
4560            BEQ YNEXTEND
```

```
4570            ;
4580            LDA YKOPIE
4590            SEC
4600            SBC OFFSET2
4610            STA YKOPIE
4620            BCS YNEXTEND
4630            DEC YKOPIE+1
4640YNEXTEND    RTS
4650            ;************************
4660            ;
4670            ;
4680            ;
4690            ;***X UND Y VERGLEICHEN***
4700XYVERGL     LDA X+1
4710            CMP Y+1
4720            BCC XYRTS
4730            BNE XYRTS               ;AUSGANG:
4740            LDA X                   ;CARRY SET, WENN
4750            CMP Y                   ;X ECHT(!) > Y
4760            BCC XYRTS
4770            BEQ XYCLC
4780XYRTS       RTS
4790XYCLC       CLC
4800            RTS
4810            ;*************************
4820            ;
4830            ;
4840            ;
4850            ;***REALZAHLEN VERGLEICHEN***
4860REALVERGL   LDA X                   ;ZEIGER AUF
4870            LDY X+1                 ;KONSTANTE A(X)
4880            JSR FACKONST            ;A(X) MIT FAC
4890            BEQ REALY               ;(=A(VG)) VERGL.
4900            BMI REALY
4910REAX        JSR XNEXT               ;X=X+1, WENN
4920            JMP REALVERGL           ;FAC<A(X)
4930            ;
4940REALY       LDA Y                   ;ZEIGER AUF
4950            LDY Y+1                 ;KONSTANTE A(Y)
4960            JSR FACKONST            ;A(Y) MIT FAC
4970            BPL REALRTS             ;(=A(VG)) VERGL.
4980REAY        JSR YLAST
4990            JMP REALY               ;Y=Y-1, WENN
5000REALRTS     RTS                     ;FAC>A(Y)
5010            ;
5020            ;
5030            ;***VERGLEICH INITIALISIEREN***
5040COMPARE     LDA TYP
5050            BEQ STRVERGL
5060            BMI REALVERGL
5070            ;
5080            ;
5090            ;***INTEGERVERGLEICH***
5100INTX        LDY #0
5110            LDA (X),Y               ;HI (X%)
5120            JSR BIT7
5130            BCC INT1                ;HI(X%)<HI(VG%) =>
5140            BNE INTY                ;GLEICH? NEIN =>
5150            ;
5160            INY
5170            LDA (X),Y               ;LO (X%)
5180            CMP DESCR1+1            ;LO (VG%)
5190            BCS INTY                ;LO(X%)>=LO(VG%)=>
5200INT1        JSR XNEXT               ;X=X+1
5210            JMP INTX
5220            ;
5230INTY        LDY #0
5240            LDA (Y),Y               ;HI (Y%)
5250            JSR BIT7
5260            BCC VINTEND             ;HI(Y%)<HI(VG%) =>
5270            BNE INT2                ;GLEICH? NEIN =>
5280            ;
5290            INY
5300            LDA (Y),Y               ;LO (Y%)
5310            CMP DESCR1+1            ;LO (VG%)
5320            BCC VINTEND             ;LO(Y%)<LO(VG%) =>
5330            BEQ VINTEND             ;GLEICH? JA =>
5340INT2        JSR YLAST               ;Y=Y-1
5350            JMP INTY
5360VINTEND     RTS
5370            ;
5380            ;
5390            ;**HIGH-BYTES VERGLEICHEN**
5400BIT7        LDX DESCR1
5410            CPX #128                ;VG% NEGATIV?
5420            BCS VGMINUS             ;JA =>
5430            CMP #128                ;X%/Y% NEGAT.?
5440            BCS NURXYMIN            ;JA =>
5450XYMITVG     CMP DESCR1              ;X%/Y% MIT VG%
5460            RTS                     ;VERGLEICHEN
5470            ;
5480VGMINUS     CMP #128                ;AUCH X%/Y%
5490            BCC NURVGMIN            ;NEGATIV? NO=>
5500            BCS XYMITVG             ;JA =>
5510            ;
5520NURXYMIN    LDA #1                  ;LOESCH.ZEROF.
5530            CLC                     ;X%/Y% ECHT(!)
5540            RTS                     ;< ALS VG%
5550            ;
5560NURVGMIN    SEC                     ;X%/Y% ECHT(!)
5570            RTS                     ;> ALS VG%
5580            ;***********************
5590            ;
5600            ;
5610            ;
5620            ;***STRINGVERGL.INITIALISIEREN***
5630STRVERGL    LDA X                   ;A$(X) MIT A$(VG)
5640VLINKS      LDX X+1                 ;VERGLEICHEN:
5650            STA STR                 ;ZEIGER AUF A$(X)
5660            STX STR+1               ;NACH STR,STR+1
5670            JSR VERGLEICH           ;VERGLEICHEN =>
5680            BCS VRE                 ;SEC = STR$>=VG$
5690            JSR XNEXT
5700            JMP STRVERGL            ;WENN STR$<VG$
5710            ;
5720VRE         LDA Y                   ;STR$(Y)<VG$?
5730VRECHTS     LDX Y+1
5740            STA STR                 ;ZEIGER AUF A$(Y)
5750            STX STR+1               ;NACH STR,STR+1
5760            JSR VERGLEICH           ;VERGLEICHEN:
5770            BCC VLI                 ;SEC = STR$>=VG$
5780            BEQ VLI
5790            JSR YLAST               ;WENN A$(Y)<VG$
5800            JMP VRE                 ;DANN Y AUF LAST
5810VLI         RTS
5820            ;
5830            ;
5840            ;***STRINGVERGLEICH***
5850VERGLEICH   LDY #2                  ;DESCRIPTOREN
5860UEBERN      LDA (STR),Y             ;VON STR$ NACH
5870            STA DESCR1,Y            ;DESCR1 BIS
5880            DEY                     ;DESCR2+2 HOLEN
5890            BPL UEBERN
5900            ;             ;DESCR AUF GEWUENSCH-
5910            JSR STRTEIL             ;TES STRINGTEILFELD
5920            LDY #255                ;INITIALISIEREN
5930            ;
5940            LDX DESCR1              ;KLEINER.STRING-
5950            CPX DESCR2              ;LAENGE INS
5960            BCC VERG                ;X-REG. UND
5970            LDX DESCR2              ;DIESES ALS
5980VERG        INX                     ;SCHLEIFENZ.VERW
5990            ;
6000VERGL       DEX                     ;VERGLEICH
6010            BNE VERGL1              ;DURCHFUEHREN
6020            LDA DESCR1
6030            CMP DESCR2
6040            RTS
6050VERGL1      INY
6060            LDA (DESCR1+1),Y        ;CARRY CLAER=
6070            CMP (DESCR2+1),Y        ;STR1<STR2
6080            BEQ VERGL               ;CARRY SET=
6090            RTS                     ;STR1>STR2
6100            ;****************************
6110            ;
6120            ;
6130            ;
6140            ;***VARIABLEN HOLEN***
6150HOLVAR      JSR CHKKOM              ;ERSTES ARRAY-
6160            JSR GETPOS              ;ELEMENT HOLEN
6170            STA LG2
6180            STY LG2+1
6190            ;
6200            JSR CHKKOM              ;LETZTES ARRAY-
6210            JSR GETPOS              ;ELEMENT HOLEN
6220            STA RG2
6230            STY RG2+1
6240            ;
6250            ;
6260            ;***TYPFLAG/OFFSET ERZEUGEN***
6270            LDX #0                  ;TYP=TYPFLAG
6280            LDY #2                  ;  0=STRING
6290            LDA STRTYP              ;127=INTEGER
6300            BNE TY                  ;128=REAL
6310            LDX #128
6320            LDY #4                  ;ZAHL=SCHL.ZAEHLER
6330            LDA NUMTYP              ;  2=STRING
6340            BEQ TY                  ;  1=INTEGER
6350            DEX                     ;  4=REAL
6360            LDY #1
6370TY          STX TYP2
6380            STY ZAHL2
6390            INY                     ;OFFSET=SCHLEIFEN-
6400            STY OFFSET2             ;ZAEHLER+1
6410            RTS
6420            ;**************************
6430            ;
6440            ;
6450            ;
6460            ;**ZEIGER AUF STRINGTEIL HOLEN**
6470            ;(AENDERT DESCR - DESCR+2:
6480            ;DESCR=LAENGE BIS NEXTFELD
6490            ;DESCR+1,DESCR+2=ZEIGER AUF
6500            ;POS.DES STRINGTEILFELDES)
6510            ;
6520STRTEIL     LDA FELD                ;NORM-VERGLEICH?
6530            BEQ TEILRTS             ;JA =>
6540            ;
6550            ;
6560            ;*ZU GEWUENSCHTEM FELD VORTASTEN*
6570            LDX FELD
6580            LDY #255
6590            ;
6600T2          DEX                     ;GEWUENSCHTES FELD
6610            BEQ TEILOK              ;GEFUNDEN? JA =>
6620            ;
6630T1          INY                     ;STRINGENDE
6640            CPY DESCR1              ;ERREICHT?
6650            BCS LNULL               ;JA =>
6660            LDA (DESCR1+1),Y
6670            CMP TRENN               ;TRENNZEICHEN?
6680            BNE T1                  ;NEIN =>
6690            BEQ T2                  ;JA =>
6700            ;
6710TEILOK      STY HELP                ;HELP=POS. 1.ZEICH
6720            INC HELP                ;V.GEWUENSCHT.FELD
6730            ;
6740            ;
6750            ;*ZU FELDENDE VORTASTEN*
6760OK1         INY                     ;TRENN=LETZTES
6770            CPY DESCR1              ;STRINGZEICHEN?
6780            BCS NEXTOK              ;JA =>
6790            LDA (DESCR1+1),Y
6800            CMP TRENN               ;TRENNEICHEN?
6810            BNE OK1                 ;NEIN =>
6820            ;
6830            ;
6840            ;*DESCRIPTOREN AUF GEW.FELD*
6850NEXTOK      TYA                     ;FELDLAENGE=
6860            SEC                     ;ANFANG NEXTFELD-
6870            SBC HELP                ;ANFANG AKT.FELD
6880            STA DESCR1
6890            ;
6900            LDA HELP                ;FELDPOSITION=
6910            CLC                     ;STRINGPOSITION+
6920            ADC DESCR1+1            ;POS.DES FELDES
6930            STA DESCR1+1            ;IM STRING
6940            BCC TEILRTS
6950            INC DESCR1+2
6960TEILRTS     RTS
6970            ;
6980LNULL       LDA #0                  ;FELDLAENGE 0,WENN
6990            STA DESCR1              ;TRENNZEICHEN =
7000            RTS                     ;LAST STRINGZEICH.
7010            ;                  BZW. GEW.FELD
7020            ;                  NICHT VORHANDEN
7030            ;******************************
7040            .EN
```

Listing 4. Der Source-Code von Quicksort wurde mit dem MAE erstellt, kann aber auf jedes beliebige andere Format angepaßt werden

Hinter den Kulissen

Routinen in Hypra-Basic einzubinden ist kein Problem, wenn diese an Hypra-Basic angepaßt sind. Mit ein paar Tricks können auch andere Programme eingebunden werden. Dazu stellen wir Ihnen nützliche Routinen aus Hypra-Basic ausführlich vor.

Einfache und kurze Routinen sind das Lebenselixier von Hypra-Basic. Damit das Programm eingebunden werden kann, sind nur einige wenige Punkte zu beachten.

Sollen keine Parameter übergeben werden (Listing 1 und Listing 5 als MSE-Lader), ist eigentlich nicht viel falsch zu machen. Listing 1 ersetzt nur ein Paar POKEs, die den Bildschirm an- und abschalten und einen C128 in den 2-MHz-Modus versetzen.

Nun die Syntax der Parameterübergabe. In Listing 2 (eine einfache Routine zum Setzen der Farben, Listing 6 ist der MSE-Lader dazu) sehen Sie den einfachsten Fall. Der erste Parameter wird ohne vorangehende Komma-Abfrage geholt, da beim Aufruf durch einen Befehlsnamen ein Komma hinter dem Namen unnötig wäre (etwa: FARBE,1). Richtig ist FARBE 1. Alle weiteren Parameter können durch Komma getrennt werden. Also FARBE 0,14,6. Bei Listing 2 wird als Routine zum Holen der Parameter $B79E genommen. Der Parameter muß zwischen 0 und 255 liegen und wird im X-Register übergeben.

Zwei weitere Tricks sind im Listing 3 (Eine MERGE-Routine, Listing 7 ist der MSE-Lader dazu) verborgen. Durch die geschickte Ausnutzung einer Betriebssystemroutine (ab Adresse $C075) spart man sich das Schreiben eines eigenen Programmteils. Zum zweiten wird ein variabler Einsprungspunkt erzeugt. Dazu wird am Programmende der nicht verwendete Sprung »JMP EMA« eingesetzt. Beim Verschieben des Programmes durch Hypra-Basic wird die Adresse EMA umgerechnet (ausgelöst durch den JMP). Das erlaubt, die Adresse hinter EMA direkt zu Laden ($C02F), um sie als Zeiger auf einen mitverschobenen Programmabschnitt zu verwenden. Mit dieser Technik sind auch Programme in den Interrupt einzubauen, wie zum Beispiel die Joysticksteuerung aus Ausgabe 7/86.

Das Problem einer formatierten Ein- und Ausgabe wird durch den Basic-V2-Interpreter leider nicht gelöst. Das Window-Modul im Hypra-Basic befriedigt nahezu alle Ansprüche. Die Ein- und Ausgabe kann in Windows definiert werden. Es beinhaltet eine umfangreiche Einlese-Routine, berücksichtigt bei der Eingabe die Cursor-Steuerung und die Funktionstasten. Ausgaben formatiert das Programm in frei definierbare Windows. Sollten Sie das Programm optimieren wollen, können Sie dies anhand des Source-Code (Listing 4 und Listing 8) leicht tun.

(J. Stellig/F. Gräf/og)

```
HYPRA-ASS  ASSEMBLERLISTING:

;********************************
;******                    ******
;***      FAST / SLOW         ***
;******                    ******
;********************************
;
; FUER DEN C-128 IM 64'ER-MODUS.
; 3/86 BY F.GRAEF, PLANKSTADT
;
              1090 -.BA $C000
;
;      --- FAST ---
;
C000 AD11D0 :1130 -FAST         LDA 53265       ;SCR AUS
C003 29EF   :1140 -             AND #239
C005 8D11D0 :1150 -             STA 53265
;
C008 A901   :1170 -             LDA #1          ;2 MHZ
C00A 8D30D0 :1180 -             STA 53296
;
C00D 60     :1200 -             RTS
;
;      --- SLOW ---
;
C00E A900   :1240 -SLOW         LDA #0          ;1 MHZ
C010 8D30D0 :1250 -             STA 53296
;
C013 AD11D0 :1270 -             LDA 53265       ;SCRN AN
C016 0910   :1280 -             ORA #16
C018 8D11D0 :1290 -             STA 53265
;
C01B 60     :1310 -             RTS
```

Listing 1. Ohne Parameter-Übergabe

```
JSR $B79E   ;GETBYTE
STX 646     ;SCHRIFT

JSR $AEFD   ;CHKCOM
JSR $B79E   ;GETBYTE
STX 53281   ;HINTERGRUND

JSR $AEFD   ;CHKCOM
JSR $B79E   ;GETBYTE
STX 53280   ;RAHMEN
RTS
```

Listing 2. Einfache Parameter

```
C000                 *=   $C000
C000         BANFL   =    $2B      ;ZEIGER AUF BASIC-
C000         BANFH   =    $2C      ;PROGRAMM-START
C000         BVARL   =    $2D      ;ZEIGER AUF START
C000         BVARH   =    $2E      ;DER VARIABLEN
C000         POL     =    $F7      ;ZEIGER IN EINZU-
C000         POH     =    $F8      ;FUEGENDES PROGRAMM
C000         STAT    =    $90      ;STATUSWORT ST
C000         CPUP    =    $01      ;PROZESSORPORT
C000         ZNR     =    $12      ;BASIC-ZEILENNUMMER
C000         SA      =    $B9      ;SEKUNDAERADRESSE
C000         FPMOD   =    $3A      ;FLAG F. PRG-MODUS
              ;
C000         ILLDI   =    $B3AB    ;ILLEGAL DIR. ERROR
C000         LPARA   =    $E1D4    ;LOAD-PARAM. HOLEN
C000         BLOAD   =    $FFD5    ;LOAD-ROUTINE D. BS
C000         FAUSW   =    $E1D1    ;FEHLERAUSWERTUNG
C000         LOERR   =    $E19C    ;LOAD - ERROR
C000         EINF    =    $A4A2    ;BASIC-Z. EINFUEGEN
C000         VEKS    =    $E455    ;BASIC-VEKTOREN SETZEN
C000         SETBP   =    $E1A7    ;BASIC-PAR. NEU SETZEN
C000         EWSL    =    $0302    ;ZEIGER AUF
C000         EWSH    =    $0303    ;EINGABE-WARTESCHLEIFE
C000         BEP     =    $01FC    ;BASIC-EINGABE-PUFFER
              ;
              ;
              ;     MERGE A - ANHAENGEN
              ;
C000 20 75 C0 AMERGE  JSR  READPAR  ;PARAMETER LESEN
C003 A5 2B            LDA  BANFL
C005 48               PHA           ;ZEIGER AUF BASIC-
C006 A5 2C            LDA  BANFH
C008 48               PHA           ;PRG-START RETTEN
C009 A5 2D            LDA  BVARL    ;ZEIGER AUF BASIC-PRG-
C00B A4 2E            LDY  BVARH    ;START HINTER BASIC-PRG
C00D 38               SEC           ;SETZEN (NEUER PRG-START =
C00E E9 02            SBC  #02      ;VARIABLENSTART - 2)
C010 B0 01            BCS  AMA
C012 88               DEY
C013 85 2B    AMA     STA  BANFL    ;NEUE ZEIGER
C015 84 2C            STY  BANFH    ;SPEICHERN
C017 20 7D C0         JSR  LOAD     ;BASIC-PRG LADEN
C01A 68               PLA
C01B 85 2C            STA  BANFH    ;ALTE ZEIGER AUF
C01D 68               PLA           ;BASIC-PRG-START WIEDER-
C01E 85 2B            STA  BANFL    ;HERSTELLEN
C020 4C A7 E1         JMP  SETBP    ;BASIC-PARAMETER SETZEN
              ;
              ;     MERGE E - EINFUEGEN
              ;
C023 20 75 C0 EMERGE  JSR  READPAR  ;PARAMETER LESEN
C026 A0 A0            LDY  #$A0     ;ANFANGSADRESSE FUER LOAD
C028 85 F7            STA  POL      ;& ZEIGER IN EINZUF. PRG
```

```
C02A 84 F8            STY  POH       ;AUF $A000 SETZEN
C02C 20 7D C0         JSR  LOAD      ;BASIC-PRG. LADEN
C02F AD 9A C0         LDA  POINTER+1
C032 AC 9B C0         LDY  POINTER+2
C035 8D 02 03         STA  EWSL      ;ZEIGER AUF EINGABE-WARTE-
C038 8C 03 03         STY  EWSH      ;SCHLEIFE NEU SETZEN
C03B A0 FF    EMA     LDY  #255      ;ZEIGER IN BASIC-ZEILE
C03D A5 01    EMB     LDA  CPUP
C03F 29 FE            AND  #%11111110
C041 85 01            STA  CPUP      ;BASIC-ROM AUSSCHALTEN
C043 C8               INY            ;ZEIGER IN ZEILE + 1
C044 B1 F7            LDA  (POL),Y   ;ZEICHEN AUS PRG HOLEN
C046 E6 01            INC  CPUP      ;BASIC-ROM EINSCHALTEN
C048 C0 01            CPY  #01       ;ZEIGER IN N. BASIC-Z. ?
C04A 90 F1            BCC  EMB       ;LOW-BYTE - NICHT BEACHTEN
C04C D0 0B            BNE  EMC       ;KEIN TEIL DES ZEIGERS
C04E AA               TAX            ;00 FUER PROGRAMMENDE ?
C04F D0 EC            BNE  EMB       ;<>00 - NICHT BEACHTEN
C051 A2 03            LDX  #03       ;PROGRAMMENDE - ZEIGER AUF
C053 20 55 E4         JSR  VEKS      ;E-WS WIEDERHERSTELLEN
C056 4C AB E1         JMP  SETBP+4   ;BASIC-PARAM. SETZEN
              ;
C059 C0 04    EMC     CPY  #04       ;ZEILENNUMMER ?
C05B B0 05            BCS  EMD       ;KEIN TEIL D. ZEILENNUMMER
C05D 99 12 00         STA  ZNR,Y     ;SPEICHERN
C060 90 DB            BCC  EMB       ;UNBEDINGTER SPRUNG
C062 99 FC 01 EMD     STA  BEP,Y     ;ZEICHEN IN PUFFER
C065 AA               TAX
C066 D0 D5            BNE  EMB       ;<>0 - KEIN ZEILENENDE
C068 98               TYA
C069 65 F7            ADC  POL       ;ZEIGER IN EINZUF. PRG
C06B 85 F7            STA  POL       ;AUF ANFANG DER NAECHSTEN
C06D 90 02            BCC  EME       ;PROGRAMMZEILE SETZEN
C06F E6 F8            INC  POH
C071 C8       EME     INY            ;LAENGE DER ZEILE + 1
C072 4C A2 A4         JMP  EINF      ;ZEILE EINFUEGEN
              ;
              ;    UNTERROUTINEN
              ;
              ;       EINGABE-PARAMETER HOLEN
              ;
C075 20 D4 E1 READPAR JSR  LPARA     ;LOAD-PARAMETER
C078 A9 00            LDA  #00       ;SEKUNDAERADRESSE = 0
C07A 85 B9            STA  SA        ;(LADEN AN FESTE ADRESSE)
C07C 60       RET     RTS
              ;
              ;       BASIC-PROGRAMM LADEN
              ;
C07D AA       LOAD    TAX            ;LOW-BYTE DER ADRESSE
C07E A5 01            LDA  CPUP
C080 29 FE            AND  #%11111110
C082 85 01            STA  CPUP      ;BASIC-ROM AUSSCHALTEN
C084 A9 00            LDA  #00       ;FLAG FUER LOAD
C086 20 D5 FF         JSR  BLOAD     ;LADEN
C089 E6 01            INC  CPUP      ;BASIC-ROM EINSCHALTEN
C08B 90 03            BCC  LOA       ;KEIN FEHLER
C08D 4C D1 E1         JMP  FAUSW     ;FEHLERAUSWERTUNG
C090 A5 90    LOA     LDA  STAT
C092 29 BF            AND  #$BF      ;STATUS UEBERPRUEFEN
C094 F0 E6            BEQ  RET       ;KEIN FEHLER
C096 4C 9C E1         JMP  LOERR
              ;
              ;       POINTER FUER HYPRA-BASIC
              ;
C099 4C 3B C0 POINTER JMP  EMA
```

Listing 3. Die MERGE-Routine nutzt das Betriebssystem

```
C000                  *=   $C000
C000          XLO     =    $F7       ;KOORDINATEN DER LINKEN
C000          YLO     =    $F8       ;OBEREN ECKE DES WINDOWS
C000          XRU     =    $F9       ;KOORDINATEN DER RECHTEN
C000          YRU     =    $FA       ;UNTEREN ECKE DES WINDOWS
C000          ZSP     =    $AA       ;ZWISCHENSPEICHER
C000          FKEY    =    $FB       ;SPEICHER F. FTASTEN
C000          CNT     =    $57       ;ZAEHLER
C000          CUX     =    $A8       ;RELATIVE CURSORPOSITION
C000          CUY     =    $A9       ;INNERHALB DES WINDOWS
C000          POL     =    $AA       ;ZWEI-BYTE ZAEHLER
C000          POH     =    $AB       ;FUER DIVERSE ZWECKE
C000          POS     =    $FC       ;POSITION IM TEXTSPEICHER
C000          ZBR     =    $FD       ;BREITE DES WINDOWS
C000          ZAN     =    $FE       ;HOEHE DES WINDOWS
C000          MPOS    =    $FF       ;LAENGE DES TEXTSPEICHERS
C000          STZ     =    $22       ;STRINGZEIGER
C000          STAT    =    $90       ;STATUS - BYTE
C000          CFLG    =    $CC       ;FLAG FUER CURSOR
C000          CREP    =    $CD       ;ZAEHLER F. CURSORBLINKEN
C000          IFLG    =    $11       ;FLAG F. INPUT,GET,READ
C000          TFLG    =    $0D       ;TYPFLAG (NUMER./STRING)
C000          CGL     =    $7A
C000          CGH     =    $7B       ;ZEIGER D. CHRGET-ROUTINE
C000          ZCGL    =    $4B       ;ZWISCHENSPEICHER FUER
C000          ZCGH    =    $4C       ;CHRGET-ZEIGER
C000          ZVL     =    $49       ;ZEIGER AUF
C000          ZVH     =    $4A       ;VARIABLENWERT
C000          COLL    =    $14
C000          COLH    =    $15       ;ZEIGER IN FARBRAM
C000          VRAM    =    $288      ;HIGH-BYTE VIDEO-RAM
C000          CCOL    =    $286      ;FARBE DES CURSORS
C000          ERLZT   =    $33C      ;TAB. F. ERLAUBTE ZEICHEN
C000          RBYTE   =    $B79E     ;1-BYTE-WERT LESEN
C000          PKOM    =    $AEFD     ;AUF KOMMA PRUEFEN
C000          PCHAR   =    $AEFF     ;AUF ZEICHEN PRUEFEN
C000          FRMEVL  =    $AD9E     ;AUSDRUCK AUWERTEN
C000          FRESTR  =    $B6A3     ;STRINGVERWALTUNG
C000          CGOT    =    $0079     ;LETZTES Z. HOLEN
C000          CGET    =    $0073     ;NAECHSTES Z. HOLEN
C000          GET     =    $FFE4     ;GET
C000          ZANFL   =    $ECF0     ;TAB. D. BILDSCHIRM-
C000          ZANFH   =    $D9       ;ZEILENANFAENGE
C000          CSET    =    $FFF0     ;CURSOR SETZEN
C000          PLET    =    $B113     ;PRUEF. AUF BUCHST.
C000          SVAR    =    $B08B     ;VARIABLE SUCHEN
C000          CGSET   =    $AB76     ;CHRGETZEIGER SETZEN
C000          STHOL   =    $B487     ;STRING HOLEN
C000          PRSET   =    $B7E2     ;PRGZEIGER SETZEN
C000          STRVAR  =    $A9DA     ;STRING AN VAR ZUW.
C000          PRBYTE  =    $B7F1     ;S.O. NACH KOMMA
C000          CRBYTE  =    $B79B     ;S.O. MIT CHRGET
              ;
              ; AUSGABE IN WINDOW
              ;
C000 20 F0 C1 PRINT   JSR  POSREAD   ;TEXT HOLEN
C003 4C 89 C1         JMP  WTEXT     ;UND IN WINDOW SCHREIBEN
              ;
              ; EINGABE IN WINDOW
              ;
              ;    - PARAMETER EINLESEN
              ;
C006 20 F0 C1 INPUT   JSR  POSREAD   ;KOORD. & TEXT
C009 A9 3C            LDA  #<ERLZT
C00B A2 03            LDX  #>ERLZT
C00D 20 2A C2         JSR  RSTRING   ;ERLAUBTE ZEICHEN LESEN
C010 8A               TXA
C011 D0 02            BNE  PEB       ;ALTE EINSTELLUNG
C013 91 AA            STA  (POL),Y   ;TAB MIT 0 ABSCHLIESSEN
C015 20 79 00 PEB     JSR  CGOT
C018 C9 2C            CMP  #","
C01A D0 0C            BNE  PEA
C01C 20 9B B7         JSR  CRBYTE    ;ANZAHL D. FTASTEN HOLEN
C01F E0 09            CPX  #09       ;VGL. <= 8
C021 B0 05            BCS  PEA       ;NEIN - NICHT BEACHTEN
C023 8A               TXA
C024 69 85            ADC  #133      ;IN ASCII UMRECHNEN
C026 85 FB            STA  FKEY      ;UND SPEICHERN
C028 A9 3B    PEA     LDA  #";"
C02A 20 FF AE         JSR  PCHAR     ;AUF SEMIKOLON PRUEFEN
              ;
              ;    - EINGABE VORBEREITEN
              ;
C02D A0 00    HOM     LDY  #00
C02F 84 A8            STY  CUX       ;CURSOR IN
C031 84 A9            STY  CUY       ;LINKE OBERE ECKE
C033 84 CC            STY  CFLG      ;CURSOR EIN
C035 84 FC            STY  POS       ;AN TEXTANFANG
              ;
              ;    - EINGABE
              ;
C037 20 7B C1 EIN     JSR  CUSWRT    ;TEXT SCHREIBEN
C03A 20 E4 FF EIA     JSR  GET       ;ZEICHEN VON TASTATUR
C03D F0 FB            BEQ  EIA       ;HOLEN
C03F A2 02            LDX  #02
C041 86 CD            STX  CREP      ;REPEATFUNKTION
              ;
              ;    - PRUEFUNG AUF SONDERZEICHEN
              ;
C043 C9 85            CMP  #"(F1)"
C045 90 08            BCC  E4        ;AUF FTASTE PRUEFEN
C047 C5 FB            CMP  FKEY      ;PR. ERLAUBTE FTASTE
C049 90 08            BCC  E5        ;JA - WIE RETURN
C04B C9 8D            CMP  #"(F8)"+1 ;PR. NICHT DEF. FTASTE
C04D 90 EB            BCC  EIA       ;JA - NICHT BEACHTEN
C04F C9 0D    E4      CMP  #13       ;PR. RETURN
C051 D0 03            BNE  E3        ;NICHT RETURN
C053 4C 31 C1 E5      JMP  RET       ;RETURN
C056 A6 A8    E3      LDX  CUX       ;CURSORPOSITION IN
C058 A4 A9            LDY  CUY       ;X/Y - REGISTER
C05A C9 13            CMP  # 19
C05C F0 CF            BEQ  HOM       ;HOME
C05E C9 8D            CMP  #141
C060 F0 52            BEQ  SRET      ;SHIFT-RETURN
C062 C9 1D            CMP  # 29
C064 F0 4B            BEQ  CUSR      ;CURSOR RIGHT
C066 C9 11            CMP  # 17
C068 F0 51            BEQ  CUSD      ;CURSOR DOWN
C06A C9 91            CMP  #145
C06C F0 5D            BEQ  CUSU      ;CURSOR UP
C06E C9 9D            CMP  #157
C070 F0 7D            BEQ  CUSL      ;CURSOR LEFT
C072 C9 14            CMP  # 20
C074 F0 65            BEQ  DEL       ;DELETE
C076 C9 94            CMP  #148
C078 F0 78            BEQ  INST      ;INSERT
C07A C9 93            CMP  #147
C07C F0 57            BEQ  CLR       ;CLEAR
              ;
              ;    - PRUEFUNG AUF ERLAUBTE ZEICHEN
```

Listing 4. So wird die Ein- und Ausgabe formatiert

```
                     ;
C07E A0 FF              LDY  #255      ;Y ALS ZEIGER IN TAB.
C080 C8         EIB     INY            ;ZEIGER ERHOEHEN
C081 BE 3C 03           LDX  ERLZT,Y   ;ZEICHEN AUS TAB. HOLEN
C084 F0 B4              BEQ  EIA       ;TABELLENENDE
C086 D9 3C 03           CMP  ERLZT,Y   ;PR. ERLAUBTES ZEICHEN
C089 F0 21              BEQ  EIE       ;ERLAUBTES ZEICHEN
C08B E0 85              CPX  #"(F1)"
C08D D0 08              BNE  EIC       ;AUF F1 PRUEFEN
C08F C9 30              CMP  #"0"      ;F1 - AUF ZIFFER PRUEFEN
C091 90 ED              BCC  EIB
C093 C9 3A              CMP  #"9"+1
C095 90 15              BCC  EIE       ;ZIFFER - ERLAUBT
C097 E0 86      EIC     CPX  #"(F3)"     ;AUF F3 PRUEFEN
C099 D0 05              BNE  EID
C09B 20 13 B1           JSR  PLET      ;F3 - AUF BUCHST. PRUEFEN
C09E B0 0C              BCS  EIE       ;BUCHSTABE - ERLAUBT
C0A0 E0 87      EID     CPX  #"(F5)"     ;AUF F5 PRUEFEN
C0A2 D0 DC              BNE  EIB
C0A4 C9 C1              CMP  #"♠"
C0A6 90 D8              BCC  EIB       ;F5 - AUF GESHIFT.
C0A8 C9 DB              CMP  #"♣"+1    ;BUCHSTABEN PRUEFEN
C0AA B0 D4              BCS  EIB       ;KEIN BUCHSTABE
C0AC A4 FC      EIE     LDY  POS       ;POSITION IM TEXTSP.
C0AE 99 4C C2 PTEXT     STA  TEXTT,Y   ;ZEICHEN SPEICHERN
                ;
                ;   - SONDERZEICHEN-BEHANDLUNG
                ;
C0B1 4C 06 C1 CUSR      JMP  INCU      ;CURSOR N. RECHTS
C0B4 A2 00      SRET    LDX  #00       ;CUS. AN ZEILENANFANG
C0B6 A5 FC              LDA  POS
C0B8 E5 A8              SBC  CUX       ;POS NEU BERECHNEN
C0BA 2C                 .BYTE$2C       ;NAECHSTEN BEF. AUSLASSEN
C0BB A5 FC      CUSD    LDA  POS
C0BD C8                 INY            ;CURSORZEILE+1
C0BE C4 FE              CPY  ZAN
C0C0 B0 03              BCS  CDB       ;CUY ZU GROSS
C0C2 65 FD              ADC  ZBR       ;POS NEU BERECHNEN
C0C4 24                 .BYTE$24       ;NAECHSTEN BEF. AUSLASSEN
C0C5 88         CDB     DEY            ;CURSORZEILE-1
C0C6 85 FC      CDA     STA  POS       ;POS SPEICHERN
C0C8 4C 18 C1           JMP  SCURS     ;CURSOR SETZEN
C0CB A5 FC      CUSU    LDA  POS
C0CD C0 00              CPY  #00
C0CF F0 F5              BEQ  CDA       ;CURS. IN 1. ZEILE
C0D1 E5 FD              SBC  ZBR       ;POS NEU BERECHNEN
C0D3 B0 F0              BCS  CDB       ;UNBEDINGTER SPRUNG
C0D5 20 1A C2 CLR       JSR  CLEAR     ;TEXT LOESCHEN
C0D8 4C 2D C0           JMP  HOM       ;ZUR HOME-ROUTINE
C0DB A4 FC      DEL     LDY  POS
C0DD F0 3D              BEQ  INCB      ;TEXTANFANG
C0DF B9 4C C2 DEB       LDA  TEXTT,Y
C0E2 99 4B C2           STA  TEXTT-1,Y
C0E5 C8                 INY
C0E6 D0 F7              BNE  DEB       ;TEXT VERSCHIEBEN
C0E8 A9 20              LDA  #32
C0EA A6 FF              LDX  MPOS
C0EC 9D 4B C2           STA  TEXTT-1,X  ;SPACE AN TEXTENDE
C0EF 4C 1F C1 CUSL      JMP  DECU      ;CURSOR LINKS
C0F2 A4 FF      INST    LDY  MPOS
C0F4 B9 4B C2 INA       LDA  TEXTT-1,Y
C0F7 99 4C C2           STA  TEXTT,Y
C0FA 88                 DEY
C0FB C4 FC              CPY  POS       ;TEXT VERSCHIEBEN
C0FD D0 F5              BNE  INA       ;BIS ZUR AUGENBL. POSITION
C0FF A9 20              LDA  #32
C101 99 4C C2           STA  TEXTT,Y   ;SPACE EINFUEGEN
C104 D0 16              BNE  INCB      ;UNBEDINGTER SPRUNG
                ;
                ;   - CURSORBEWEGUNGEN
                ;
C106 A6 A8      INCU    LDX  CUX       ;CURSOR RIGHT
C108 A4 A9              LDY  CUY       ;CPOS LADEN
C10A E8                 INX            ;SPALTE + 1
C10B E4 FD              CPX  ZBR
C10D D0 07              BNE  INCC      ;KEINE NEUE ZEILE
C10F A2 00              LDX  #00       ;ZEILENANFANG
C111 C8                 INY            ;ZEILE + 1
C112 C4 FE              CPY  ZAN
C114 F0 06              BEQ  INCB      ;CURSOR RIGHT UNMOEGLICH
C116 E6 FC      INCC    INC  POS       ;POS. IN TEXT + 1
C118 86 A8      SCURS   STX  CUX       ;REL. CURSORPOSITION
C11A 84 A9              STY  CUY       ;SPEICHERN
C11C 4C 37 C0 INCB      JMP  EIN       ;ZUR EINGABESCHLEIFE
C11F A4 A9      DECU    LDY  CUY       ;CURSOR LEFT
C121 A6 A8              LDX  CUX
C123 D0 06              BNE  DECB      ;NICHT AM ZEILENANFANG
C125 A6 FD              LDX  ZBR       ;ZEILENENDE
C127 98                 TYA
C128 F0 F2              BEQ  INCB      ;CURSOR LEFT UNMOEGLICH
C12A 88                 DEY            ;ZEILE - 1
C12B CA         DECB    DEX            ;SPALTE - 1
C12C C6 FC              DEC  POS       ;POS. IN TEXT - 1
C12E 4C 18 C1           JMP  SCURS
                ;
                ;   - ENDE DER EINGABE
                ;
C131 86 CC      RET     STX  CFLG      ;CURSOR AUS
C133 E9 83              SBC  #"(F1)"-2  ;FTASTENNR. ERRECHNEN
C135 B0 02              BCS  REA       ;NICHT RETURN
C137 A9 00              LDA  #00       ;00 FUER RETURN
C139 85 90      REA     STA  STAT      ;IN ST SPEICHERN
C13B 20 89 C1           JSR  WTEXT     ;TEXT AUF BS AUSGEBEN
C13E BD 4B C2 REB       LDA  TEXTT-1,X
C141 C9 20              CMP  #32
C143 D0 03              BNE  REC       ;TEXT NACH UEBER-
C145 CA                 DEX            ;FLUESSIGEN LEERZEICHEN
C146 D0 F6              BNE  REB       ;DURCHSUCHEN
C148 A9 00      REC     LDA  #00
C14A 9D 4C C2           STA  TEXTT,X   ;00 FUER TEXTENDE
C14D 85 11              STA  IFLG      ;00 FUER INPUT
C14F 20 8B B0           JSR  SVAR      ;VARIABLE SUCHEN
C152 24 0D              BIT  TFLG
C154 10 22              BPL  SYNERR    ;NUMERISCHE VARIABLE
C156 85 49              STA  ZVL       ;ZEIGER AUF VARIABLE
C158 84 4A              STY  ZVH       ;SPEICHERN
C15A A5 7A              LDA  CGL
C15C A4 7B              LDY  CGH       ;CHRGET-ZEIGER
C15E 85 4B              STA  ZCGL
C160 84 4C              STY  ZCGH      ;ZWISCHENSPEICHERN
C162 AD AF C0           LDA  PTEXT+1   ;ANFANGSADRESSE DES
C165 AC B0 C0           LDY  PTEXT+2   ;TEXTES UEBERGEBEN
C168 20 87 B4           JSR  STHOL
C16B 20 E2 B7           JSR  PRSET     ;STRING HOLEN
C16E 20 DA A9           JSR  STRVAR    ;UND DER VAR. ZUWEISEN
C171 A5 4B              LDA  ZCGL
C173 A4 4C              LDY  ZCGH      ;CHRGET-ZEIGER
C175 4C 76 AB           JMP  CGSET     ;WIEDERHERSTELLEN
                ;
C178 4C 08 AF SYNERR    JMP  $AF08     ;SYNTAX ERROR
                ;
                ;   - UNTERROUTINEN
                ;
                ;       - CURSOR SETZEN
                ;
C17B 18         CUSWRT  CLC
C17C A5 A9              LDA  CUY       ;AUS RELATIVER
C17E 65 F8              ADC  YLO
C180 AA                 TAX            ;CURSORPOSITION
C181 A5 A8              LDA  CUX       ;ABSOLUTE CURSORPOSITION
C183 65 F7              ADC  XLO
C185 A8                 TAY            ;BERECHNEN
C186 20 F0 FF           JSR  CSET      ;CURSOR SETZEN
                ;
                ;       - TEXT IN WINDOW SCHREIBEN
                ;
C189 A2 00      WTEXT   LDX  #00       ;ZEIGER IN TEXT
C18B A5 F8              LDA  YLO
C18D 85 57              STA  CNT       ;ZEILENZAEHLER
C18F A4 57      WTA     LDY  CNT       ;BS-ZEILE
C191 B9 F0 EC           LDA  ZANFL,Y   ;ZEIGER IN
C194 85 AA              STA  POL       ;BILDSCHIRMSPEICHER
C196 85 14              STA  COLL      ;UND FARBRAM SETZEN
C198 B9 D9 00           LDA  ZANFH,Y
C19B 29 03              AND  #03
C19D 48                 PHA
C19E 0D 88 02           ORA  VRAM
C1A1 85 AB              STA  POH
C1A3 68                 PLA
C1A4 09 D8              ORA  #$D8
C1A6 85 15              STA  COLH
C1A8 A4 F7              LDY  XLO       ;SPALTEN - ZAEHLER
C1AA BD 4C C2 WTB       LDA  TEXTT,X   ;ZEICHEN HOLEN
C1AD 30 08      BCWAND  BMI  BCA       ;UMWANDLUNG ASCII-
C1AF C9 60              CMP  #96
C1B1 90 04              BCC  BCA       ;CODE IN
C1B3 29 DF              AND  #%11011111  ;BILDSCHIRMCODE
C1B5 D0 02              BNE  BCB
C1B7 29 BF      BCA     AND  #%10111111
C1B9 10 02      BCB     BPL  BCE
C1BB 49 C0              EOR  #%11000000
C1BD 91 AA      BCE     STA  (POL),Y   ;ZEICHEN IN BSRAM
C1BF AD 86 02           LDA  CCOL      ;SCHREIBEN UND
C1C2 91 14              STA  (COLL),Y  ;FARBE SETZEN
C1C4 E8                 INX            ;ZEIGER IN TEXT + 1
C1C5 C8                 INY            ;SPALTENZAEHLER + 1
C1C6 C4 F9              CPY  XRU
C1C8 90 E0              BCC  WTB       ;NOCH NICHT LETZTE SPALTE
C1CA E6 57              INC  CNT       ;ZEILENZAEHLER + 1
C1CC E4 FF              CPX  MPOS      ;PR. AUF TEXTENDE
C1CE 90 BF              BCC  WTA       ;NOCH NICHT TEXTENDE
C1D0 60                 RTS
                ;
                ;       - KOORDINATENPAAR LESEN
                ;
C1D1 20 FD AE RKOORD    JSR  PKOM      ;AUF KOMMA PRUEFEN
C1D4 86 AA              STX  ZSP       ;ZAEHLER FUER KOORDINATEN
C1D6 20 9E B7           JSR  RBYTE     ;BYTE-WERT LESEN
C1D9 E0 28              CPX  #40       ;PR. < 40 (X-KOORDINATE)
C1DB B0 10              BCS  ILLERR    ;NEIN - FEHLERMELDUNG
C1DD A4 AA              LDY  ZSP
C1DF 96 F7              STX  XLO,Y     ;WERT SPEICHERN
C1E1 20 F1 B7           JSR  PRBYTE    ;BYTE-WERT LESEN
C1E4 E0 19              CPX  #25       ;PR. < 25 (Y-K.)
C1E6 B0 05              BCS  ILLERR    ;NEIN
C1E8 A4 AA              LDY  ZSP
C1EA 96 F8              STX  XLO+1,Y   ;SPEICHERN
C1EC 60                 RTS
                ;
C1ED 4C 48 B2 ILLERR    JMP  $B248     ;ILLEGAL Q. ERROR
                ;
                ;       - WINDOWPARAMETER & TEXT-
                ;         STRING LESEN
                ;
C1F0 A2 00      POSREAD LDX  #00       ;1.KOORDINATENPAAR
C1F2 20 D4 C1           JSR  RKOORD+3  ;LESEN
C1F5 A2 02              LDX  #02
C1F7 20 D1 C1           JSR  RKOORD    ;2.  ''
C1FA E6 F9              INC  XRU       ;(BESSER VERARBEITBAR)
C1FC 38                 SEC
C1FD A5 F9              LDA  XRU       ;BREITE DES WINDOWS +1
C1FF E5 F7              SBC  XLO       ;BERECHNEN
C201 90 EA              BCC  ILLERR    ;<0 - FEHLER
C203 85 FD              STA  ZBR       ;UND SPEICHERN
C205 8A                 TXA            ;HOEHE DES WINDOWS + 1
C206 E5 F8              SBC  YLO       ;BERECHNEN
C208 90 E3              BCC  ILLERR
C20A 69 00              ADC  #00
C20C 85 FE              STA  ZAN
C20E AA                 TAX            ;HOEHE (IN X-REG.) MIT
C20F A9 00              LDA  #00       ;BREITE MALNEHMEN = LAENGE
C211 65 FD      POA     ADC  ZBR       ;DES EINGABETEXTES
C213 B0 D8              BCS  ILLERR    ;>255 - FEHLER
C215 CA                 DEX
C216 D0 F9              BNE  POA
```

```
C218 85 FF            STA  MPOS      ;LAENGE SPEICHERN
C21A A0 00    CLEAR   LDY  #00       ;TEXTSPEICHER
C21C A9 20            LDA  #32       ;LOESCHEN
C21E 99 4C C2 CLEA    STA  TEXTT,Y
C221 C8               INY
C222 D0 FA            BNE  CLEA
               ;
               ;    - STRING LESEN & SPEICHERN
               ;
C224 AD AF C0 RSTRINGT LDA PTEXT+1   ;ZEIGER AUF
C227 AE B0 C0         LDX  PTEXT+2   ;TEXTSPEICHER
C22A 85 AA    RSTRING STA  POL       ;ZEIGER FUER
C22C 86 AB            STX  POH       ;SPEICHERUNG D. STRINGS
C22E 20 79 00         JSR  CGOT      ;LETZTES ZEICHEN HOLEN
C231 C9 2C            CMP  #","      ;PR. AUF KOMMA
C233 D0 16            BNE  RSTE      ;NEIN - KEIN STRING FOLGT
C235 20 73 00         JSR  CGET
C238 20 9E AD         JSR  FRMEVL    ;STRING LESEN
C23B 20 A3 B6         JSR  FRESTR    ;& PARAMETER HOLEN
C23E A0 00            LDY  #00
C240 AA               TAX            ;LAENGE D. STRING IN X-REG.
C241 F0 08            BEQ  RSTE      ;LAENGE = 0
C243 B1 22    RSTB    LDA  (STZ),Y   ;STRING IN
C245 91 AA            STA  (POL),Y   ;SPEICHER (POL/H =
C247 C8               INY            ;ANFANGSADRESSE) VERSCHIEBEN
C248 CA               DEX
C249 D0 F8            BNE  RSTB
C24B 60       RSTE    RTS
              ;
              ;    - TEXTSPEICHER
              ;
C24C          TEXTT   =    *
```

Listing 4. Formatierte Ein- und Ausgabe (Schluß)

```
Name : fast/slow                c000 c01c
-----------------------------------------
c000 : ad 11 d0 29 ef 8d 11 d0   e0
c008 : a9 01 8d 30 d0 60 a9 00   52
c010 : 8d 30 d0 ad 11 d0 09 10   7b
c018 : 8d 11 d0 60 20 9b b7 e0   ed
```

Listing 5. »FAST«, wie in Listing 1

```
Name : farbe                    c000 c019
-----------------------------------------
c000 : 20 9e b7 8e 86 02 20 fd   24
c008 : ae 20 9e b7 8e 21 d0 20   da
c010 : fd ae 20 9e b7 8e 20 d0   52
c018 : 60 11 d0 60 20 9b b7 e0   c0
```

Listing 6. »FARBE« mit einfachen Parametern

```
Name : merge                    c000 c09c
-----------------------------------------
c000 : 20 75 c0 a5 2b 48 a5 2c   a3
c008 : 48 a5 2d a4 2e 38 e9 02   53
c010 : b0 01 88 85 2b 84 2c 20   db
c018 : 7d c0 68 85 2c 68 85 2b   33
c020 : 4c a7 e1 20 75 c0 a0 a0   dd
c028 : 85 f7 84 f8 20 7d c0 ad   35
c030 : 9a c0 ac 9b c0 8d 02 03   4f
c038 : 8c 03 03 a0 ff a5 01 29   9e
c040 : fe 85 01 c8 b1 f7 e6 01   d3
c048 : c0 01 90 f1 d0 0b aa d0   9d
c050 : ec a2 03 20 55 e4 4c ab   57
c058 : e1 c0 04 b0 05 99 12 00   16
c060 : 90 db 99 fc 01 aa d0 d5   38
c068 : 98 65 f7 85 f7 90 02 e6   3b
c070 : f8 c8 4c a2 a4 20 d4 e1   96
c078 : a9 00 85 b9 60 aa a5 01   ae
c080 : 29 fe 85 01 a9 00 20 d5   71
c088 : ff e6 01 90 03 4c d1 e1   ea
c090 : a5 90 29 bf f0 e6 4c 9c   70
c098 : e1 4c 3b c0 13 b1 b0 0c   20
```

Listing 7. »MERGE« aus Ausgabe 9/86

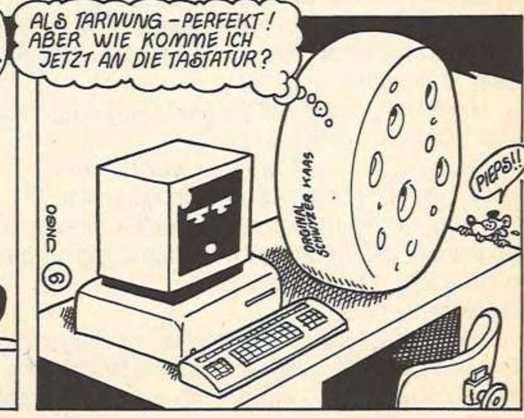
```
Name : window                   c000 c24c
-----------------------------------------
c000 : 20 f0 c1 4c 89 c1 20 f0   9b
c008 : c1 a9 3c a2 03 20 2a c2   61
c010 : 8a d0 02 91 aa 20 79 00   47
c018 : c9 2c d0 0c 20 9b b7 e0   2c
c020 : 09 b0 05 8a 69 85 85 fb   e5
c028 : a9 3b 20 ff ae a0 00 84   70
c030 : a8 84 a9 84 cc 84 fc 20   3a
c038 : 7b c1 20 e4 ff f0 fb a2   f5
c040 : 02 86 cd c9 85 90 08 c5   ba
c048 : fb 90 08 c9 8d 90 eb c9   67
c050 : 0d d0 03 4c 31 c1 a6 a8   1d
c058 : a4 a9 c9 13 f0 cf c9 8d   75
c060 : f0 52 c9 1d f0 4b c9 11   42
c068 : f0 51 c9 91 f0 5d c9 9d   02
c070 : f0 7d c9 14 f0 65 c9 94   9e
c078 : f0 78 c9 93 f0 57 a0 ff   d5
c080 : c8 be 3c 03 f0 b4 d9 3c   ab
c088 : 03 f0 21 e0 85 d0 08 c9   fa
c090 : 30 90 ed c9 3a 90 15 e0   fb
c098 : 86 d0 05 20 13 b1 b0 0c   65
c0a0 : e0 87 d0 dc c9 c1 90 d8   b2
c0a8 : c9 db b0 d4 a4 fc 99 4c   57
c0b0 : c2 4c 06 c1 a2 00 a5 fc   0d
c0b8 : e5 a8 2c a5 fc c8 c4 fe   d8
c0c0 : b0 03 65 fd 24 [illegible] 85 fc   a1
c0c8 : 4c 18 c1 a5 fc c0 00 f0   fd
c0d0 : f5 e5 fd b0 f0 20 1a c2   4b
c0d8 : 4c 2d c0 a4 fc f0 3d b9   3f
c0e0 : 4c c2 99 4b c2 c8 d0 f7   03
c0e8 : a9 20 a6 ff 9d 4b c2 4c   23
c0f0 : 1f c1 a4 ff b9 4b c2 99   4d
c0f8 : 4c c2 88 c4 fc d0 f5 a9   e1
c100 : 20 99 4c c2 d0 16 a6 a8   02
c108 : a4 a9 e8 e4 fd d0 07 a2   1f
c110 : 00 c8 c4 fe f0 06 e6 fc   5a
c118 : 86 a8 84 a9 4c 37 c0 a4   13
c120 : a9 a6 a8 d0 06 a6 fd 98   1f
c128 : f0 f2 88 ca c6 fc 4c 18   c2
c130 : c1 86 cc e9 83 b0 02 a9   be
c138 : 00 85 90 20 89 c1 bd 4b   57
c140 : c2 c9 20 d0 03 ca d0 f6   c1
c148 : a9 00 9d 4c c2 85 11 20   bf
c150 : 8b b0 24 0d 10 22 85 49   99
c158 : 84 4a a5 7a a4 7b 85 4b   8d
c160 : 84 4c ad af c0 ac b0 c0   21
c168 : 20 87 b4 20 e2 b7 20 da   9f
c170 : a9 a5 4b a4 4c 4c 76 ab   ac
c178 : 4c 08 af 18 a5 a9 65 f8   e6
c180 : aa a5 a8 65 f7 a8 20 f0   fb
c188 : ff a2 00 a5 f8 85 57 a4   ef
c190 : 57 b9 f0 ec 85 aa 85 14   89
c198 : b9 d9 00 29 03 48 0d 88   1b
c1a0 : 02 85 ab 68 09 d8 85 15   f4
c1a8 : a4 f7 bd 4c c2 30 08 c9   a2
c1b0 : 60 90 04 29 df d0 02 29   5d
c1b8 : bf 10 02 49 c0 91 aa ad   c8
c1c0 : 86 02 91 14 e8 c8 c4 f9   0a
c1c8 : 90 e0 e6 57 e4 ff 90 bf   7d
c1d0 : 60 20 fd ae 86 aa 20 9e   11
c1d8 : b7 e0 28 b0 10 a4 aa 96   1d
c1e0 : f7 20 f1 b7 e0 19 b0 05   fe
c1e8 : a4 aa 96 f8 60 4c 48 b2   95
c1f0 : a2 00 20 d4 c1 a2 02 20   ae
c1f8 : d1 c1 e6 f9 38 a5 f9 e5   07
c200 : f7 90 ea 85 fd 8a e5 f8   68
c208 : 90 e3 69 00 85 fe aa a9   32
c210 : 00 65 fd b0 d8 ca d0 f9   73
c218 : 85 ff a0 00 a9 20 99 4c   60
c220 : c2 c8 d0 fa ad af c0 ae   92
c228 : b0 c0 85 aa 86 ab 20 79   28
c230 : 00 c9 2c d0 16 20 73 00   6a
c238 : 20 9e ad 20 a3 b6 a0 00   89
c240 : aa f0 08 b1 22 91 aa c8   86
c248 : ca d0 f8 60 00 00 00 c0   46
```

Listing 8. »WINDOW« aus Ausgabe 9/86

Listing 5 bis 8 können Sie mit dem MSE eingeben







Das Feinste vom Feinen

In einem Sonderheft mit dem Thema »Assembler« darf natürlich eine Programmsammlung mit ausgewählten Maschinenroutinen nicht fehlen. Wir haben für Sie unter anderem ein Programm, das laufend die aktuelle Adresse von Maschinenroutinen auf dem Bildschirm zeigt, und drei geniale Einzeiler ausgesucht.

Bitte beachten Sie beim Eintippen unserer Listings unbedingt unsere Eingabehinweise auf Seite 92. Die Maschinensprache-Listings können Sie auch mit einem normalen Maschinensprache-Monitor eingeben. Dabei lassen Sie in jeder Zeile das letzte Byte (=Prüfsumme) weg.

OLD-Funktion für Variablen

Dieses Programm (Listing 1) holt nicht nur ein Basic-Programm nach NEW, CLR oder einem Reset zurück, sondern auch sämtliche Variablen, Strings und Felder.

```
programm : re-clr                   c000 c073
---------------------------------------------
c000 : a0 01 98 91 2b 20 33 a5   45
c008 : 18 a5 22 69 02 85 2d a5   f5
c010 : 23 69 00 85 2e 20 60 a6   4b
c018 : a0 00 b1 2f aa 29 7f 20   3d
c020 : 13 b1 90 49 8a 0a b0 10   35
c028 : c8 b1 2f 10 0b a0 05 b1   c4
c030 : 2f d0 1b c8 b1 2f d0 16   ab
c038 : a0 07 e6 2f d0 02 e6 30   14
c040 : 88 d0 f7 a5 2f 85 31 a5   12
c048 : 30 85 32 4c 18 c0 a0 00   5b
c050 : b1 31 29 7f 20 13 b1 90   57
c058 : 14 c8 c8 18 b1 31 65 31   a2
c060 : aa c8 b1 31 65 32 85 32   63
c068 : 86 31 4c 4e c0 20 26 b5   75
c070 : 4c ae a7 00 bf bf 1e 01   72
```

Listing 1. »RE-CLR«. Verwenden Sie zur Eingabe bitte den MSE (Seite 92).

```
LABEL      LOC    CODE        STATEMENT
STPRG      0000               *=   $C000
Old        C000   A0 01       LDY  #01
           C002   98          TYA
           C003   91 2B       STA  (2B),Y   eine »NEW-Null« löschen (mit 1 überschreiben)
           C005   20 33 A5    JSR  A533     Basic-Zeilen neu binden
           C008   18          CLC
           C009   A5 22       LDA  22
           C00B   69 02       ADC  #02
           C00D   85 2D       STA  2D
           C00F   A5 23       LDA  23       Programmende (= Zeiger +2 Nullen) überlesen
           C011   69 00       ADC  #00
           C013   85 2E       STA  2E
           C015   20 60 A6    JSR  A660     zum CLR-Befehl
-----------------------------------------------------------------------------
Variablen  C018   A0 00       LDY  #00
           C01A   B1 2F       LDA  (2F),Y   1. Byte nach Programmende
           C01C   AA          TAX           retten
           C01D   29 7F       AND  #7F      Bit 7 löschen
           C01F   20 13 B1    JSR  B113     Buchstabe?
           C022   90 49       BCC  C06D     nein, denn fertig
           C024   8A          TXA
           C025   0A          ASL           war Bit 7 gesetzt?
           C026   B0 10       BCS  C038     ja, kein String (sondern Integervariable oder Funktion),
           C028   C8          INY           Arrayerkennungsschleife überspringen
           C029   B1 2F       LDA  (2F),Y   2. Namensbyte
           C02B   10 0B       BPL  C038     kleiner als $80, also REAL-Variable, kein String
           C02D   A0 05       LDY  #05
           C02F   B1 2F       LDA  (2F),Y   6. Byte des Strings
           C031   D0 1B       BNE  C04E     ungleich Null => Array-Feldkopf, kein String
           C033   C8          INY
           C034   B1 2F       LDA  (2F),Y   7. Byte des Strings
           C036   D0 16       BNE  C04E     ungleich Null => Array-Feldkopf, kein String
           C038   A0 07       LDY  #07
           C03A   E6 2F       INC  2F
           C03C   D0 02       BNE  C040
           C03E   E6 30       INC  30       7 Bytes überlesen
           C040   88          DEY
           C041   D0 F7       BNE  C03A
           C043   A5 2F       LDA  2F
           C045   85 31       STA  31
           C047   A5 30       LDA  30       Array-Ende = Variablen-Ende
           C049   85 32       STA  32
           C04B   4C 18 C0    JMP  C018     und weitermachen; nächstes Element prüfen
-----------------------------------------------------------------------------
ARRAY      C04E   A0 00       LDY  #00
           C050   B1 31       LDA  (31),Y   1. Byte nach Variablenende
           C052   29 7F       AND  #7F      Bit 7 löschen
           C054   20 13 B1    JSR  B113     Buchstabe?
           C057   90 14       BCC  C06D     nein, dann fertig: keine Felder mehr
           C059   C8          INY
           C05A   C8          INY
           C05B   18          CLC
           C05C   B1 31       LDA  (31),Y   Feldlänge Low-Byte
           C05E   65 31       ADC  31       zu Array-Ende-Zeiger addieren und merken
           C060   AA          TAX
           C061   C8          INY
           C062   B1 31       LDA  (31),Y   Feldlänge High-Byte
           C064   65 32       ADC  32       zu Array-Zeiger addieren
           C066   85 32       STA  32       Feld überlesen und
           C068   86 31       STX  31       Array-Zeiger auf Feld-Ende positionieren
           C06A   4C 4E C0    JMP  C04E     und weitermachen
-----------------------------------------------------------------------------
Ende       C06D   20 26 B5    JSR  B526     Garbage-Collection: Stringzeiger nachstellen
           C070   4C AE A7    JMP  A7AE     zurück zur Interpreterschleife
```

Bild 1. Der dokumentierte Quelltext von Listing 1

Das vorliegende Maschinenprogramm steht ab $C000 im Speicher (kann verschoben werden), benötigt 115 Byte Speicherplatz und wird einfach mit »SYS 49152« aufgerufen.

Es können, wie gesagt, alle Werte und sogar definierte Funktionen und Felder zurückgeholt werden. Allerdings bestehen da noch drei kleine Bedingungen:

1. Es muß vorher bereits ein Basic-Programm im Speicher gestanden haben (mindestens: »10 :«).
2. Es darf vor Aufrufen der Routine kein Variablen-Programm eingegeben oder ein Feld abgefragt worden sein.
3. Die erste Dimensionierung eines Feldes muß ein Stringfeld sein. Beispiele:

- »DIM DE(20),DE$(20)« muß umgestellt werden zu »DIM DE$(20),DE(20)«
- »DIM E(34)« muß ergänzt werden zu »DIM A$(0),E(34)« (A$(0) genügt schon!!)

Diese Ergänzung braucht der Computer nämlich, um zu erkennen, daß in die Array-Behandlung gesprungen werden muß, sonst entsteht ein Fehler.

Funktionsweise

Der Computer untersucht das erste Byte nach dem Basic-Programm-Ende. Wenn dieses ein Buchstabe ist, prüft er es auf Stringvariable. Wenn nicht, wird der Zähler einfach um 7 Byte (zwei Namensbyte und fünf Informationsbyte) erhöht und diese übergangen.

Wenn aber eine Stringvariable gefunden wurde, untersucht er Byte 6 und 7. Diese sind bei einem String nämlich immer Null, weil sie hier nicht gebraucht werden; bei einem Feldkopf hingegen ist die Anzahl der Elemente der Dimensionierung des folgenden Feldes darin abgelegt: Wenn der Computer also eine numerische als erste Dimensionierung findet, hält er sie für eine normale numerische Variable (denn eine andere Erkennung ist hier leider nicht möglich!) und nicht für den ersten Feldkopf der Arrays, gibt einen falschen und unsinnigen Wert für die meist nicht vorhandene einfache Variable und findet auch die restlichen Felder nicht mehr!

Wenn keine Buchstaben mehr gefunden werden, wird noch eine Garbage-Collection durchgeführt, um die Stringzeiger wieder nachzustellen.

Beachten Sie bitte:

Falls Sie zum Beispiel eingeben

```
10 DIM A$(30),B$(10)
20 CLR
30 DIM A$(30)
40 SYS 49152
```

wird automatisch A$(30), aber auch B$(10) zurückgeholt!!

Tip: Das Programm kann auch nach einem »NEW«, »CLR« oder »Reset« eingeladen werden. Sie können das Programm auch auf anderen Speicherbereichen laufen lassen (zum Beispiel im Kassettenpuffer). Dazu müssen Sie die beiden Sprungbefehle

```
C04B JMP C018
C06A JMP C04E
```

jeweils anpassen (und die SYS-Adresse). Mit dem SMON ist das Verschieben ja kein Problem.

Übrigens: Bild 1 zeigt das dokumentierte Assembler-Listing von »RE-CLR«. (Andreas Blödow/tr)

Die Super-Einzeiler

Wer nicht will, daß sein Basic-Programm unnötig viel Speicher verbraucht, da er jedes freie Byte für Daten gebrauchen kann, oder wer will, daß sein Programm möglichst schnell ist, weil ihm jede Sekunde kostbar ist, für den ist es oft von Nutzen, mehrere Programmzeilen zu einer zusammenzufassen. Jede Zeile, die neu begonnen wird, verbraucht nämlich vier Byte mehr Speicherplatz, als wenn diese mit einem Doppelpunkt an die vorhergehende Zeile angehängt würde. Bei einem längeren Programm kann da schon einiges zusammenkommen. Außerdem kann der Interpreter Befehle in einer Zeile schneller abarbeiten, als wenn jeder Befehl in einer eigenen Zeile steht. Wenn diese sich dann noch in einer Schleife befinden, die vielleicht einige hundertmal durchlaufen wird, könnte man hier schon durch eine andere Anordnung Zeit sparen. Um das Programm allerdings nicht allzu unübersichtlich zu machen, sollte man nur solche Zeilen zusammenfassen, die auch logisch zusammengehören.

Wenn man dies nun beherzigen will, wird man allerdings durch den Basic-Editor ziemlich eingeschränkt, da dieser nur maximal 80 Zeichen in einer Zeile zuläßt. Der Interpreter würde aber auch längere Zeilen akzeptieren, nur kann man solche eben nicht eingeben.

Um dieses Manko zu beseitigen, habe ich EX-LINE geschrieben (siehe Listing 2). Mit diesem Programm ist es nun möglich, Basic-Zeilen mit einer Länge von bis zu 252 Zeichen einzugeben.

```
programm : ex-line              c000 c12e
-----------------------------------------

c000 : a0 00 a2 a0 84 f7 86 f8   71
c008 : a2 20 b1 f7 91 f7 c8 d0   c3
c010 : f9 e6 f8 e6 fa ca d0 f2   c6
c018 : a9 c2 8d f5 a4 8d 84 a5   58
c020 : 8d ba a5 8d e7 a5 8d 06   13
c028 : a6 a9 c1 8d 13 a5 8d 16   86
c030 : a5 8d 24 a5 8d cd a5 8d   53
c038 : d0 a5 8d f1 a5 8d 0b a6   bd
c040 : 8d d2 aa 8d fd c1 a2 87   1a
c048 : a0 c0 8e 08 03 8c 09 03   ac
c050 : a9 d0 a2 7a 8d 28 a5 8e   27
c058 : 2b a5 8c 2c a5 a9 20 a2   6c
c060 : 82 8d 80 a4 8e 81 a4 8c   fe
c068 : 82 a4 a2 00 bd 1d c1 f0   93
c070 : 06 20 d2 ff e8 d0 f5 6c   01
c078 : 02 a0 b9 fc c1 91 5f 4c   97
c080 : 59 a6 a9 37 85 01 60 20   a0
c088 : 73 00 c9 21 f0 03 4c e7   ba
c090 : a7 a2 36 86 01 20 73 00   c6
c098 : b0 21 a2 ae a0 c0 8e 00   ai
c0a0 : 03 8c 01 03 48 a9 93 20   ea
c0a8 : d2 ff 68 4c a4 a6 20 ea   f3
c0b0 : e8 a2 8b a0 e3 8e 00 03   99
c0b8 : 8c 01 03 a9 13 20 d2 ff   38
c0c0 : 20 cf ff ad 00 04 48 a2   04
c0c8 : 00 86 02 a5 d4 48 a9 00   77
c0d0 : 85 d4 a9 13 20 d2 ff a9   78
c0d8 : 27 85 d0 bd 00 04 8d 00   04
c0e0 : 04 68 85 d4 20 cf ff 9d   d0
c0e8 : 00 c2 18 65 02 85 02 e8   22
c0f0 : d0 d9 8a 8d ff c2 a2 fc   9c
c0f8 : bd 00 c2 c9 20 d0 04 ca   cd
c100 : b8 50 f5 e8 a9 00 9d 00   8c
c108 : c2 68 8d 00 04 a9 0d a2   69
c110 : 03 20 d2 ff ca d0 fa 20   37
c118 : cf aa 4c 86 a4 0d 45 58   99
c120 : 2d 4c 49 4e 45 20 42 59   a1
c128 : 20 48 43 45 0d 00 ff ff   b7
```

Listing 2. Die Basic-Erweiterung »EX-LINE«

Nachdem man dieses Programm mit dem MSE eingegeben oder mit »LOAD ''EX-LINE'',8,1« von Diskette geladen hat, kann man es mit »SYS 49152« starten. Danach steht das Ausrufungszeichen als neuer Befehl zur Verfügung. Wenn man diesen Befehl eingibt, erscheint der Cursor in der linken oberen Ecke des Bildschirms. Nun kann man eine Programmzeile oder Befehle im Direktmodus eingeben. Dabei ist zu beachten, daß nach <RETURN> nur die ersten 6,5 Bildschirmzeilen übernommen werden, egal wo sich der Cursor dann befindet. Wenn man nach dem Ausrufungszeichen noch eine Zeilennummer angibt, wird vorher noch der Bildschirm gelöscht und die entsprechende Zeile gelistet, so daß man auch überlange Zeilen editieren kann.

Da sich das Programm nicht mit dem Checksummer verträgt, errechnet es noch eine Prüfsumme, die sich mit »PRINT PEEK(2)« auslesen läßt. Sie entsteht einfach durch Addition der ASC-Werte der eingegebenen Zeichen. In der Speicherzelle 2 steht dann das Low-Byte der Summe.

Wenn man die Befehle in abgekürzter Form eingibt, erhält man eine andere Prüfsumme, als wenn man sie ausschreibt. Die angegebenen Prüfsummen bei den folgenden Programmen beziehen sich immer auf die ausgeschriebene Form.

Programmbeschreibung

Das Programm ist komplett in Maschinensprache geschrieben und steht im Speicher ab $C000. Nach dem Start mit »SYS 49152« wird zuerst das Basic-ROM in den darunter liegenden RAM-Bereich kopiert und so verändert, daß der Eingabepuffer nicht mehr bei $0200, sondern bei $C200 liegt. Dies ist nötig, da dieser jetzt mehr Platz benötigt, als das Betriebssystem für ihn vorsieht.

Schließlich wird der Vektor für »Basic-Befehl holen« bei $0308 auf die neue Routine gelenkt. Diese prüft, ob das erste Zeichen ein Ausrufungszeichen ist. Wenn dies der Fall ist und noch eine Zeilennummer folgt, wird zunächst der Bildschirm gelöscht und die entsprechende Zeile gelistet. (Wenn keine Zeile angegeben war, wird dieser Teil übersprungen.)

Nachdem dann die Eingabe erfolgt ist, wird das RAM bei $A000 aktiviert, der Text im oberen Teil des Bildschirms in den Puffer bei $C200 kopiert, in Interpretercode umgewandelt und ausgeführt. Das ROM wird bei der nächsten Eingabe wieder eingeschaltet, so daß der Eingabepuffer wieder wie üblich bei $0200 liegt.

Um zu zeigen, was man mit diesem Programm alles in einer Zeile unterbringen kann, habe ich folgende drei »Einzeiler« geschrieben:

```
1 OPEN 1,8,15:FOR I=0 TO 44:READ A:PRINT#1
  ,"M-W"CHR$(I)CHR$(5)CHR$(1)CHR$(A):NEXT:
  PRINT#1,"UC":DATA 169,254,170,32,21,5,20
  2,224,1,208,248,32,21,5,232,224,255,208,
  248,240,235,138,72,73,255,168,169,248,14
  1,0,28,202,208,248,169,240,141,,28,136,2
  08,248,104,170,96
```

© 64'er

Listing 3. »Soft Flash« läßt die rote LED der Floppy stufenlos an- und ausgehen. Zur Eingabe Listing 2 verwenden.

1. SOFT FLASH:

Dieser Einzeiler (Listing 3) bewirkt, daß die rote LED am Diskettenlaufwerk scheinbar stufenlos ein- und ausgeschaltet wird. Wenn man das Programm mit Hilfe von »EX-LINE« eingegeben hat, kann man mit »PRINT PEEK(2)« die Prüfsumme abfragen. Diese sollte 180 betragen.

Programmbeschreibung:

In der FOR-NEXT-Schleife wird ein Maschinenprogramm mittels »Memory-Write« in das RAM der Floppy ab $0500 geschrieben. Der Floppy-Befehl »UC« bewirkt dann, daß dieses gestartet wird. Das Maschinenprogramm schaltet die LED so schnell an und aus, daß dies für das Auge nicht sichtbar ist. Dabei ändert sich die Länge der Hell- und Dunkel-

phase, so daß es scheint, als ob die Lampe langsam hell und dunkel würde.

Weil das Programm als Endlosschleife geschrieben ist, läßt sich das Laufwerk nicht mehr ansprechen. Wenn man die Floppy weiter verwenden will, muß man sie vorher aus- und anschalten.

2. STRICH-CURSOR:

Dieser Einzeiler (Listing 4) verwandelt den Cursor in einen Strich, der unter den Zeichen blinkt, so wie man es oft bei größeren Computern sieht. Man kann diese Routine verwenden, um seinen Programmen ein professionelles Aussehen zu verleihen (Prüfsumme: 186).

Programmbeschreibung:

Die ersten drei POKE-Befehle bewirken, daß der Bildschirm nach $CC00 verlegt wird und der Zeichensatz aus dem RAM ab $D000 gelesen wird. Dies ist notwendig, da kein Basic-Speicher verlorengehen soll und der VIC nur 16 KByte auf einmal adressieren kann.

Bevor man nun den Originalzeichensatz bei $D000 mit »POKE 1,3« lesbar machen kann, muß noch der Interrupt mit »POKE 56333,1« ausgeschaltet werden. Die erste FOR-NEXT-

```
1 W=56333:Q=53248:Z=4↑5:POKE Q+24,52:POKE
   56576,0:POKE 648,204:POKE W,1:POKE 1,3:F
   OR I=0 TO 999:POKE 52224+I,PEEK(Z+I):NEX
   T:FOR I=0 TO Z:A=Q+I:B=A+2*Z:L=Z*((I AND
    7)=7):POKE A,PEEK(A):POKE B,PEEK(B):POK
   E A+Z,PEEK(A-L):POKE B+Z,PEEK(B-L):NEXT:
   POKE 1,7:POKE W,129
```

© 64'er

Listing 4. »Strich-Cursor«. Verleiht Ihrem Cursor ein äußerst professionelles Aussehen.

```
1 PRINT"{HOME,RVSON,SPACE,RVOFF}*{RVSON}9{
   RVOFF}CD{RVSON}Y{RVOFF},A{RVSON}H{RVOFF}
   P{RVSON}Y{RVOFF}U{RVSON}){RVOFF}3{RVSON}
   E{RVOFF}A{RVSON}){RVOFF}@{RVSON}EY)PEU{S
   PACE,RVOFF}@{RVSON}1Y.9#{RVOFF}J&B{RVSON
   }JP@X{RVOFF}IG{RVSON}(%{RVOFF}B{RVSON}QY
   X{RVOFF}IG{RVSON}(HP‡‡U%UI{SHIFT-SPACE}P
   ↑){RVOFF}7{RVSON}E{RVOFF}AL,AU{RVSON})DM
   {RVOFF}X{RVSON}P){RVOFF}@{RVSON}M{RVOFF}
   @{RVSON}-){RVOFF}C{RVSON}M{RVOFF}TC{RVSO
   N}){RVOFF}A{RVSON}M{RVOFF}UCX{SHIFT-SPAC
   E,RVSON}){RVOFF}@{RVSON}*(H={RVOFF}@D{RV
   SON}Y‡T={RVOFF}@E{RVSON}Y‡T={RVOFF}@F{RV
   SON}Y‡{SHIFT-SPACE}={RVOFF}@G{RVSON}Y‡*H
   ‡PB{RVOFF}L1{RVSON}N{RVOFF}":SYS 1024
```

© 64'er

Listing 5. »Upside-Down«. Dreht den Bildschirm um 180 Grad! Beachten Sie bitte die Eingabehinweise im Text.

Schleife kopiert den Bildschirm an seine neue Position. Die zweite Schleife kopiert den Zeichensatz ins RAM. Dabei werden die reversen Zeichen so verändert, daß nur die unterste Reihe revers erscheint. Die folgenden POKEs bewirken schließlich, daß der Interrupt eingeschaltet wird und der I/O-Bereich bei $D000 wieder ansprechbar ist.

3. UPSIDE-DOWN:

Dieses Programm habe ich in zwei Versionen geschrieben, da es als Einzeiler (Listing 5) sehr schwer abzutippen ist. Die zweite Version (Listing 6) wird mit dem MSE eingegeben (die Prüfsumme für den Einzeiler lautet 76). Beide Versionen können einfach mit »RUN« gestartet werden. Danach erscheint der ganze Bildschirm »auf den Kopf gestellt«. Dies bezieht sich auf *alle* Ein- und Ausgaben. Auch Zeichen, die mit

```
programm : upside-down v2  0801 0896
-------------------------------------

0801 : 11 08 00 00 9e 32 30 37  c1
0809 : 32 20 ab 20 48 43 45 00  ee
0811 : 00 00 00 00 00 00 00 a0  53
0819 : 40 b9 5b 08 99 2c 01 88  1e
0821 : 10 f7 78 a9 33 85 01 a9  37
0829 : 00 85 f7 a9 d0 85 f8 a0  7d
0831 : 00 b1 f7 ae b9 a3 4a 26  0c
0839 : 02 ca d0 fa 98 49 07 a8  75
0841 : a5 02 91 f7 98 49 07 a8  8c
0849 : c8 d0 e6 e6 f8 a5 f8 c9  44
0851 : e0 d0 de a9 37 85 01 4c  c2
0859 : 2c 01 78 a9 84 8d 18 d0  10
0861 : a9 00 8d 00 dd a9 43 8d  c1
0869 : 14 03 a9 01 8d 15 03 58  c8
0871 : 60 a9 00 aa a8 88 bd 00  c1
0879 : 04 99 e8 e2 bd 00 05 99  03
0881 : e8 e1 bd 00 06 99 e8 e0  5c
0889 : bd 00 07 99 e8 df 88 e8  bd
0891 : d0 e4 4c 31 ea 20 8e b4  60
```

Listing 6. »Upside-Down V2«. Listing 5 als MSE Dump.

»POKE« auf den Bildschirm gebracht werden, erscheinen an der entsprechenden anderen Stelle auf dem Kopf. Die meisten Programme vertragen sich damit gut, da nur der IRQ-Vektor verändert wird. Nach einem Reset oder RUN/STOP-RESTORE kann das Programm mit »SYS 300« wieder aktiviert werden.

Programmbeschreibung:

Das Maschinenprogramm, das bei dem Einzeiler mittels »PRINT« auf den Bildschirm gebracht wird, kopiert zunächst eine Interruptroutine in den Stack ab Adresse 300. (Dieser Bereich ist besonders geeignet, da er nach einem Reset nicht gelöscht wird und ein Programm dort nicht stört.) Dann wird der gesamte Zeichensatz auf den Kopf gestellt und in das RAM bei $D000 kopiert. Der VIC wird veranlaßt, den Zeichensatz aus $D000 und den Bildschirm aus $E000 zu lesen, und schließlich wird die Interruptroutine aktiviert, die 60mal in der Sekunde den Bildschirm umgekehrt nach $E000 kopiert.

(H. C. Edelmann/tr)

Computer-Logbuch

Dieses Assemblerprogramm (Listing 7) dient dazu, die aktuelle Speicherstelle im Programmlauf in die linke obere Ecke des Bildschirms zu schreiben.

Ursprünglich wurde bei jedem zehnten Interrupt die Adresse aus den Speicherstellen 105 und 106 gelesen und in die oberste Zeile des Bildschirms geschrieben.

Durch den schnellen Wechsel der Anzeige war dies jedoch mühsam abzulesen. Aus diesem Grund habe ich jede zweite

```
programm : outadr              c000 c097
----------------------------------------

c000 : 78 a9 0d a0 c0 8d 14 03  73
c008 : 8c 15 03 58 60 ce 96 c0  43
c010 : ad 96 c0 c9 0f f0 11 ad  8a
c018 : 96 c0 c9 0a f0 25 ad 96  de
c020 : c0 c9 01 f0 39 4c 31 ea  b4
c028 : a0 03 a9 0e 99 00 d8 88  84
c030 : 10 f8 a0 00 bd 06 01 20  35
c038 : 66 c0 bd 05 01 20 66 c0  3b
c040 : 4c 31 ea a0 03 a9 0e 99  dc
c048 : 28 d8 88 10 f8 a0 00 bd  10
c050 : 06 01 20 7e c0 bd 05 01  bf
c058 : 20 7e c0 4c 31 ea a9 10  a2
c060 : 8d 96 c0 4c 31 ea 48 4a  12
c068 : 4a 4a 4a 20 71 c0 68 29  7f
c070 : 0f c9 0a 90 02 e9 39 69  20
c078 : 30 99 00 04 c8 60 48 4a  3b
c080 : 4a 4a 4a 20 89 c0 68 29  19
c088 : 0f c9 0a 90 02 e9 39 69  38
c090 : 30 99 28 04 c8 60 10 18  17
```

Listing 7. »Outadr« zeigt ständig den Prozessor-Programmzähler an

Ausgabe in die zweite Zeile umgeleitet. Nun ist die Ablesung kein Problem mehr. Es ist schon interessant, bei dieser Adressenfolge zuzuschauen. Besonders lehrreich ist es für Anfänger und Fortgeschrittene, zu sehen, welche Bereiche im Kernel angesprungen werden.

Denkbar wäre noch, die Ablesung bei jedem Interrupt vorzunehmen, oder die Adressen auf den Drucker zu geben. Dabei könnten alle Adressen im ROM-Bereich aussortiert werden, wenn man sich auf die Struktur eines noch unbekannten Programms konzentrieren möchte. (Ralf Störmer/tr)

Der Super-Autostart

Darauf haben Sie schon lange gewartet: Einen Autostart-Generator, der viele sinnvolle Eigenschaften aufweist. Dazu gehören: Kurzes Listing (sowohl des Generator-Programms als auch des Autostarts selber), einfach in der Anwendung, RUN/STOP-RESTORE- und Reset-Schutz für das fertige Programm und eine eingebaute Codier- und Decodier-Funktion.

Der Autostart-Generator in Listing 8 hat alle genannten Funktionen. Die Anwendung ist äußerst einfach: Abtippen, speichern, absolut laden, »NEW« eintippen. Dann das zu bearbeitende (Basic-)Programm laden und den Autostart mit folgender Zeile aktivieren:

```
SYS 49152,Code,"Haupt-Name","Lader-Name"
```

»Code« ist eine beliebige Zahl zwischen 0 und 255. »Haupt-Name« und »Lader-Name« sind die zukünftigen Namen des codierten Hauptteils beziehungsweise des Lade-Programms auf der Diskette. Der Lader ist später mit »,8,1« in den C 64 zu lesen.

Läßt man »,"Lader-Name"« weg, so wird nur der codierte Hauptteil neu gespeichert (wenn man Änderungen am Hauptteil vorgenommen hat). Wichtig ist dann nur, daß die Code-Zahl des Hauptprogramms mit der des Laders übereinstimmt. Im Zweifelsfall sollte man lieber den alten Lader löschen und beide Teile neu generieren.

(Christoph Dautzenberg/tr)

```
programm : autostart            c000 c131
-----------------------------------------
c000 : 20 fd ae 20 9e b7 8e ac   0a
c008 : c0 20 fd ae 20 9e ad 20   1b
c010 : a3 b6 8d 6f c0 c9 0d 90   0f
c018 : 03 4c 71 a5 20 bd ff 20   83
c020 : 11 c1 a2 08 86 ba 20 35   e5
c028 : c0 a9 2b a6 2d a4 2e 20   4d
c030 : d8 ff 4c bf c0 a5 2b 85   04
c038 : fb a5 2c 85 fc a0 00 b1   fa
c040 : fb 4d ac c0 91 fb c8 d0   e3
c048 : f6 a5 fc e6 fc c5 2e d0   85
c050 : ee 60 a2 ea 8e 28 03 bd   26
c058 : 77 02 4d 00 03 9d 80 7f   42
c060 : ca 30 f4 a2 04 bd 10 fd   3e
c068 : 9d 04 80 ca 10 f7 a9 0c   00
c070 : a2 0d a0 80 20 bd ff a9   14
c078 : 00 85 9d 20 d5 ff 86 2d   78
c080 : 98 a6 2b 86 fb a4 2c 84   a6
c088 : fc 20 57 a6 a8 b1 fb 4d   e2
c090 : 00 03 91 fb c8 d0 f6 a5   30
c098 : 2e e7 fc 10 f0 20 53 e4   22
c0a0 : 4c ae a7 e2 fc 5e fe 43   cf
c0a8 : 48 44 38 36 00 45 a6 02   b0
c0b0 : 28 43 29 38 36 20 42 59   eb
c0b8 : 20 43 48 44 4c 79 00 20   e5
c0c0 : 35 c0 20 79 00 c9 2c d0   2d
c0c8 : f3 20 73 00 20 9e ad 20   96
c0d0 : a3 b6 c9 00 d0 05 a2 08   11
c0d8 : 4c 37 a4 20 bd ff a2 52   f8
c0e0 : a0 c0 86 ac 84 ad a2 bc   d1
c0e8 : a0 c0 86 ae 84 af a9 61   8f
c0f0 : 85 b9 20 d5 f3 20 8f f6   81
c0f8 : a9 08 20 b1 ff a9 61 20   f7
c100 : 93 ff a9 a6 20 dd ed a9   ce
c108 : 02 20 dd ed a0 00 4c 24   d3
c110 : f6 86 fb 84 fc a8 88 b1   73
c118 : fb 4d ac c0 99 b0 c0 88   30
c120 : 10 f5 a0 03 b9 a3 c0 4d   0a
c128 : ac c0 99 a3 c0 88 10 f4   8a
c130 : 60 ff ff ff ff ff ff ff   90
```

Listing 8. Der Super-Autostart, den Sie schon immer suchten

Fehlersuche für Einsteiger

Vor allem Anfänger haben Schwierigkeiten, die manchmal nur schwer verständlichen englischen Fehlermeldungen des C64 richtig zu deuten. Aber gerade während der ersten Schritte in Basic ist es wichtig zu wissen, was man denn eigentlich falsch gemacht hat. Dieses Programm (Listing 9) übersetzt für Sie die Meldungen des Basic-Interpreters und bringt die fehlerhafte Programmzeile gleich auf den Bildschirm. Nach der Eingabe mit dem MSE speichern Sie das Listing erst einmal auf Diskette oder Kassette. Laden Sie es dann absolut, also mit LOAD"FEHLER",8,1 für Diskette beziehungsweise mit LOAD"FEHLER",1,1 für Datasette. Starten Sie dann das Programm mit SYS 49152. Wenn Sie die Erweiterung nicht mehr brauchen, geben Sie einfach SYS 49163 ein.

(Florian Gudermann/tr)

```
programm : fehler              c000 c2d8
----------------------------------------
c000 : a9 16 8d 00 03 a9 c0 8d   b3
c008 : 01 03 60 a9 8b 8d 00 03   03
c010 : a9 e3 8d 01 03 60 8a 10   ac
c018 : 03 4c 74 a4 0a aa bd 99   13
c020 : c2 85 fa bd 9a c2 85 fb   e9
c028 : 20 cc ff a9 00 85 13 20   9c
c030 : d7 aa a9 62 a0 c0 20 1e   e0
c038 : ab a5 fa a4 fb 20 1e ab   99
c040 : a4 3a c8 f0 03 20 c2 bd   09
c048 : a9 0d 20 16 e7 ea ea a5   0f
c050 : 39 85 14 a5 3a 85 15 20   6a
c058 : a7 a6 a9 80 20 90 ff 4c   ec
c060 : 80 a4 11 12 12 12 2a 2a   68
c068 : 2a 20 20 46 45 48 4c 45   c6
c070 : 52 20 3a 20 20 2a 2a 2a   b5
c078 : 92 0d 9a 00 00 00 5a 55   4c
c080 : 20 56 49 45 4c 45 20 46   c2
c088 : 49 4c 45 53 00 46 49 4c   a3
c090 : 45 20 4f 46 46 45 4e 00   4a
c098 : 46 49 4c 45 20 4e 49 43   5f
c0a0 : 48 54 20 4f 46 46 45 4e   4d
c0a8 : 00 46 49 4c 45 20 4e 49   c8
c0b0 : 43 48 54 20 47 45 46 55   93
c0b8 : 4e 44 45 4e 00 47 45 52   37
c0c0 : 45 41 54 20 4e 49 43 48   8c
c0c8 : 54 20 56 4f 52 48 41 4e   b5
c0d0 : 44 45 4e 00 4b 45 49 4e   eb
c0d8 : 20 45 49 4e 47 41 42 45   c9
c0e0 : 46 49 4c 45 00 4b 45 49   89
c0e8 : 4e 20 41 55 53 47 41 42   3a
c0f0 : 45 46 49 4c 45 00 46 49   34
c0f8 : 4c 45 4e 41 4d 45 46 45   45
c100 : 48 4c 54 00 55 4e 45 52   05
c108 : 4c 41 55 42 54 45 20 47   11
c110 : 45 52 41 45 54 4e 55 4d   1f
c118 : 4d 45 52 00 4e 45 58 54   b6
c120 : 20 4f 48 4e 45 20 46 4f   d1
c128 : 52 00 54 45 58 54 46 45   04
c130 : 48 4c 45 52 00 52 45 54   8a
c138 : 55 52 4e 20 4f 48 4e 45   49
c140 : 20 47 4f 53 55 42 00 46   36
c148 : 45 48 4c 45 4e 44 45 20   c9
c150 : 44 41 54 41 00 55 4e 45   e1
c158 : 52 4c 41 55 42 54 45 52   4c
c160 : 20 42 45 52 45 49 43 48   79
c168 : 00 45 52 47 45 42 4e 49   bb
c170 : 53 20 5a 55 20 47 52 4f   39
c178 : 53 53 00 53 50 45 49 43   ba
c180 : 48 45 52 20 56 4f 4c 4c   ad
c188 : 00 45 58 49 53 54 49 45   f2
c190 : 52 54 20 4e 49 43 48 54   57
c198 : 00 4e 49 43 48 54 44 49   45
c1a0 : 4d 45 4e 53 49 4f 4e 49   69
c1a8 : 45 52 54 00 5a 57 45 49   33
c1b0 : 4d 41 4c 20 44 49 4d 45   03
c1b8 : 4e 53 49 4f 4e 49 45 52   d5
c1c0 : 54 00 44 55 52 43 48 20   71
c1c8 : 4e 55 4c 4c 20 47 45 54   57
c1d0 : 45 49 4c 54 00 44 49 52   43
c1d8 : 45 4b 54 4d 4f 44 55 53   95
c1e0 : 20 4e 49 43 48 54 20 45   14
c1e8 : 52 4c 41 55 42 54 00 53   c9
c1f0 : 54 52 49 4e 47 20 3c 3d   6a
c1f8 : 3e 20 5a 41 48 4c 56 45   d0
c200 : 52 57 45 43 48 53 45 4c   84
c208 : 54 00 53 54 52 49 4e 47   f3
c210 : 20 5a 55 20 4c 41 4e 47   4d
c218 : 00 46 41 4c 53 43 48 45   10
c220 : 20 44 41 54 45 4e 20 45   0f
c228 : 49 4e 45 53 20 46 49 4c   46
c230 : 45 53 00 5a 55 20 4b 4f   8c
c238 : 4d 50 4c 49 5a 49 45 52   93
c240 : 54 45 52 20 41 55 53 44   64
c248 : 52 55 43 4b 00 4b 45 49   81
c250 : 4e 20 43 4f 4e 54 20 4d   0c
c258 : 4f 45 47 4c 49 43 48 00   75
c260 : 46 45 48 4c 45 4e 44 45   47
c268 : 20 44 45 46 20 41 4e 57   b8
c270 : 45 49 53 55 4e 47 00 56   a5
c278 : 45 52 49 46 59 20 50 52   7e
c280 : 47 2e 00 4c 4f 41 44 20   b8
c288 : 50 52 2e 00 21 55 4e 54   2c
c290 : 45 52 42 52 45 43 48 55   13
c298 : 4e 47 00 7e c0 8d c0 98   06
c2a0 : c0 a9 c0 bd c0 d4 c0 e5   9e
c2a8 : c0 f6 c0 04 c1 1c c1 2a   ec
c2b0 : c1 35 c1 47 c1 55 c1 69   06
c2b8 : c1 7b c1 89 c1 99 c1 4c   61
c2c0 : c1 c2 c1 d5 c1 ef c1 0a   c4
c2c8 : c2 19 c2 33 c2 4d c2 60   90
c2d0 : c2 77 c2 83 c2 8c c2 00   0a
```

Listing 9. »Deutsche Fehler«

Der Super-Kopierschutz

Eigentlich verliert ein guter Disketten-Kopierschutz seinen Sinn, wenn sein Mechanismus in einer Zeitschrift veröffentlicht wird. Der hier vorliegende ist jedoch so außergewöhnlich gut, daß wir ihn Ihnen nicht vorenthalten wollten. Nebenbei erfahren Sie eine Menge über den internen Aufbau und die Funktionsweise des 1541-Laufwerks.

In den meisten Büchern wird der Aufbau einer Diskette symmetrisch dargestellt. Diese Annahme ist aber falsch (zumindest im Falle einer Diskette, die mit einem 1541-Laufwerk formatiert wurde). Die tatsächliche Anordnung der Sektoren zeigt Bild 1. Die Aufgabe eines guten Kopierschutzes könnte also sein, die Verschiebung der Blöcke zweier Tracks gegeneinander zu messen. Dies ist jedoch weitaus schwieriger, als man auf den ersten Blick vermutet.

Zunächst: Warum ist diese Methode so effektiv (in bezug auf die Kopiersicherheit)? Fast alle erhältlichen Kopierprogramme können nämlich nur eines: Einen Track mit allen heutzutage bekannten Gemeinheiten (Geschwindigkeitsänderungen zwischen zwei Blöcken oder innerhalb eines Blocks, versteckte Daten im Blockheader oder provozierte Fehlermeldungen mit versteckten Informationen) mehr oder weniger identisch auf eine andere Diskette bringen. Was sie nicht können: Die Lageverschiebungen zwischen zwei Tracks mitkopieren. Ein Kopierschutz, der auf diese Weise arbeitet, müßte mit den zur Zeit bekannten Mitteln unkopierbar sein.

Ein Kopierschutz entsteht

Zuerst mußte für diese zeitkritische Aufgabe ein zuverlässiger Zeitgeber gefunden werden. Es wurde der Interrupttimer bei Adresse $1C05 des Laufwerks gewählt. Im nächsten Schritt wurde versucht, die Zeit zwischen dem Wechsel von einer Spur auf eine andere und dem Finden eines bestimmten Sektors zu messen. Nach einigen Versuchen stellte sich jedoch heraus, daß diese Methode außer perfekten Zufallszahlen keine vernünftigen Ergebnisse lieferte: Durch das automatische Lesen der Sync-Markierung und des Blockheaders beim Spurwechsel durch das Betriebssystem der 1541 wird die gemessene Zeit zu stark verfälscht.

Eine neue Möglichkeit mußte her: Der Track wird komplett mit lauter $55-Bytes (binär %01010101) beschrieben. Danach kommt auf auf eine beliebige Stelle desselben Tracks eine $FF-Markierung (binär %11111111). Auf diese Weise kann man das Lesen der Sync-Markierung beim Trackwechsel, das ja die Zeit verfälscht, ausschalten (es existieren nun ja keine Syncs mehr).

Nächster Probelauf der Entwicklungsphase: Ein bestimmter Block wird eingelesen und der Timer auf Null gestellt. Darauf erfolgt ein Wechsel auf den eben präparierten Track. Nun wird auf die $FF-Marke gewartet und die Zeit bis dahin gestoppt. Wenn diese Zeit nicht mit einem festen Wert übereinstimmt, handelt es sich nach Adam Riese um eine Kopie der Diskette.

Entscheidende Nachteile dieser Methode:

1. Auf der Diskette geht für die Programmspeicherung ein kompletter Track (nämlich der präparierte) verloren, und
2. Da jedes Diskettenlaufwerk leicht unterschiedliche Umdrehungszahlen hat, läßt sich das Programm wegen der festen Zeitvorgabe nicht übertragen.

Nächster Schritt: Ein Programm wurde entwickelt, das den Trackwechsel selbständig vornimmt. Dadurch konnte das automatische Lesen der Sync-Markierung und des Headers verhindert werden. Das Programm mißt nun die Zeit vom Lesen eines Blocks auf einer Spur bis zum Erreichen einer Sync-Markierung auf dem nächsten Track. Leider ergab sich dabei wieder ein Problem: Da ein Track mit maximal 40 Syncs bestückt sein kann, sind die gemessenen Zeiten einfach zu klein, um als Basis für eine Abfrage dienen zu können.

Was nun? Als nächster Versuch wird ein Track dadurch präpariert, daß ein beliebiger Block mit $F0-Daten-Bytes gefüllt wird (binär %11110000). Auf der Diskette erscheinen diese Werte dann aufgrund der GCR-Codierung der 1541 als $55-Bytes. Das Programm wartet nun nach dem Trackwechsel, bis entweder $55- oder $AA-Bytes ankommen. $AA deshalb, weil es vorkommen kann, daß man beim Lesen zuerst ein gesetztes Bit erwischt: $55=%01010101 und $AA=%10101010. Die Zeit wird wieder gemessen und mit einem Fixwert verglichen. Aber: Die Übertragung auf andere Laufwerke mit anderen Umdrehungsgeschwindigkeiten klappte immer noch nicht.

Die Lösung dieses Problems liegt eigentlich auf der Hand und führte schließlich zum nun vorliegenden Programm: Auf drei nebeneinanderliegenden Tracks wird je ein mit $F0-Bytes »behandelter« Block als Markierung gesetzt. Das Programm mißt nun einmalig die Zeit von Track 1 zu Track 2 und von Track 2 zu Track 3. Diese beiden Werte werden auf Diskette gespeichert. Soll nun der Kopierschutz abgefragt werden, nimmt das Programm wieder eine Messung zwi-

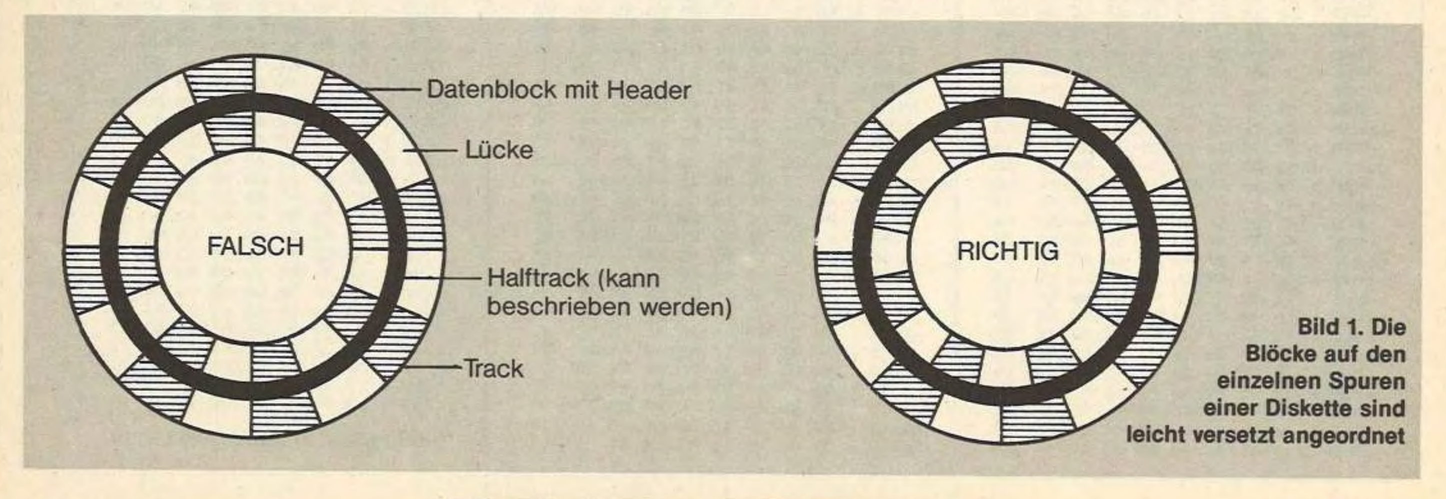


Bild 1. Die Blöcke auf den einzelnen Spuren einer Diskette sind leicht versetzt angeordnet

schen Track 1 und Track 2 und zwischen Track 2 und Track 3 vor. Die jetzt erhaltenen Werte stimmen aufgrund der unterschiedlichen Umdrehungszahlen von zwei verschiedenen Laufwerken nicht mehr überein. Überprüft werden jetzt nicht die Werte selbst, sondern das Verhältnis dieser Werte zueinander, denn das muß ja gleich sein (sofern es sich um ein Original handelt). Dabei wird eine kleine Toleranz gewährt. Erste Testläufe zeigten: Das Prinzip funktioniert hervorragend!

Am effektivsten ist Schutz bei den Tracks 3, 4, 5 und 6. Die meisten Kopierprogramme fordern nämlich an diesen Stellen zum Wechseln der Diskette auf, können folglich die Verschiebung zwischen den Tracks nicht mitbekommen.

Vorteile des Schutzverfahrens:
- Sehr hohe Wirkung
- Leicht aufzutragen
- Abfrage des Schutzes ist Schreib-/Lesekopfschonend
- Der Mechanismus läßt sich nicht so leicht aufdecken

Nachteile:
- Eventuelle Kompatibilitätsprobleme mit besonderen Laufwerken
- Speicherverlust von zirka zehn Blöcken auf der Diskette
- Auf den mit $F0-Marken versehenen Tracks darf sich kein anderer Datenblock befinden, der ebenfalls eine bestimmte Anzahl von $F0-Bytes enthält.

Bedienung der Programme

Listing 1 ist nur ein kleines Ladeprogramm für die eigentliche Abfrageroutine »protect1« (siehe Listing 2) und das Steuerprogramm »protect2« (Listing 3). Diese Programme müssen Sie mit dem MSE auf Seite 92 eintippen. Achten Sie bitte beim Eingeben der Programmnamen in den MSE auf korrekte Schreibweise. Das Ladeprogramm findet sonst die beiden anderen Programme auf der Diskette nicht.

Wenn Sie das Ladeprogramm mit LOAD"LADER PROTECT", 8 geladen und mit RUN gestartet haben, werden Sie als erstes aufgefordert, eine Diskette in das Laufwerk zu legen und eine beliebige Taste zu drücken. Auf diese Diskette wird der Schutz dann angebracht. Sie sollte möglichst noch leer (aber bereits formatiert!) sein, da eventuell vorhandene Programme gelöscht werden könnten. Nachdem Sie das Einlegen durch einen Tastendruck bestätigt haben, wird Ihnen noch eine Bedenkzeit von zehn Sekunden überlassen. Falls Sie jetzt bemerken, daß Sie aus Versehen doch die falsche Diskette eingelegt haben, können Sie den Vorgang noch mit <RUN/STOP> abbrechen. Während der Arbeit des Programms (rote Leuchtdiode an der Diskettenstation brennt) darf die Diskette unter keinen Umständen aus dem Laufwerk genommen werden! Die Mechanik könnte sonst Schaden nehmen.

In den nächsten Sekunden geschieht mit Ihrer Diskette folgendes: Die Spuren 1 bis 6 werden neu formatiert und je ein Block pro Spur bekommt eine Abfragemarke aus $F0-Bytes. Die Blocknummer dieses Blocks ist dabei gleich der Tracknummer (also zum Beispiel Track 3, Block 3). Also beschreiben Sie bei der späteren Verwendung dieser Diskette niemals einen dieser Blöcke auf den Spuren 1 bis 6!

Sind die Blöcke präpariert, wird auf Track 1, Sektor 0 noch die Kopfpositionierungsroutine und ein Abfrageprogramm unter dem Namen »C.OBJ« auf die Diskette geschrieben.

Wenn Sie nun überprüfen möchten, ob die Diskette kopiert wurde, laden Sie dieses Programm mit LOAD"C.OBJ",8,1 in den C 64 und starten es mit SYS 28672. Die Diskettenstation arbeitet einige Sekunden (rote Leuchtdiode brennt) und das Programm meldet sich wieder mit READY. Durch PRINT


```
Name : lader protect      0801 0870
-----------------------------------
0801 : 23 08 00 00 9e 20 32 30  3d
0809 : 38 38 20 20 28 43 4f 50  e4
0811 : 59 52 49 47 48 54 20 42  fb
0819 : 59 20 44 2e 49 52 49 91  c9
0821 : 29 00 00 00 00 00 00 ad  a6
0829 : 21 d0 8d 86 02 a9 01 a2  9d
0831 : 08 a0 01 20 ba ff a9 08  30
0839 : a2 5f a0 08 20 bd ff a9  f7
0841 : 00 20 d5 ff a9 01 a2 08  04
0849 : a0 01 20 ba ff a9 08 a2  7c
0851 : 67 a0 08 20 bd ff a9 00  91
0859 : 20 d5 ff 4c 10 27 50 52  0d
0861 : 4f 54 45 43 54 31 50 52  49
0869 : 4f 54 45 43 54 32 46 02  90
```

Listing 1. »LADER PROTECT«, das kurze Ladeprogramm für Listing 2 und 3

```
Name : protect1           7000 73a2
-----------------------------------
7000 : 20 01 71 20 1f 71 20 b5  6b
7008 : 70 ad a8 73 c9 05 d0 1a  24
7010 : ad e2 70 18 69 06 90 03  5d
7018 : ee e3 70 8d e2 70 20 0d  12
7020 : 71 20 b5 70 20 01 71 4c  85
7028 : 44 71 20 01 71 20 0d 71  7c
7030 : 20 22 71 60 a9 28 85 fe  ba
7038 : a9 72 85 ff 8a 48 a9 08  1d
7040 : 8d e1 71 a9 00 8d 5b 70  0a
7048 : 20 eb 70 a9 4d 20 dd ed  d8
7050 : a9 2d 20 dd ed a9 57 20  1d
7058 : dd ed a9 00 20 dd ed a9  92
7060 : 05 20 dd ed a9 1f 20 dd  7a
7068 : ed a0 00 b1 fe 20 dd ed  20
7070 : c8 c0 1f d0 f6 18 98 6d  e8
7078 : 5b 70 8d 5b 70 20 fe ed  ba
7080 : 18 a5 fe 69 1f 90 02 e6  a4
7088 : ff 85 fe ce e1 71 d0 b8  42
7090 : 20 eb 70 a9 4d 20 dd ed  20
7098 : a9 2d 20 dd ed a9 45 20  1d
70a0 : dd ed 68 20 dd ed a9 05  90
70a8 : 20 dd ed 20 fe ed ad a6  9a
70b0 : 71 8d b4 70 60 20 eb 70  bb
70b8 : a9 4d 20 dd ed a9 2d 20  ed
70c0 : dd ed a9 52 20 dd ed a9  44
70c8 : 00 20 dd ed a9 04 20 dd  04
70d0 : ed a9 07 20 dd ed 20 fe  23
70d8 : ed 20 f6 70 a0 07 20 13  8a
70e0 : ee 99 a1 73 88 d0 f7 20  a1
70e8 : fb ed 60 a9 08 20 0c ed  b5
70f0 : a9 6f 20 b9 ed 60 a9 08  29
70f8 : 20 09 ed a9 6f 20 c7 ed  40
7100 : 60 20 eb 70 a9 49 20 dd  9a
7108 : ed 20 fe ed 60 ae df 71  60
7110 : 8e 3d 72 8e ce 72 e8 8e  ec
7118 : 69 72 a2 8a 4c 34 70 a2  22
7120 : 29 2c a2 1c a9 e2 85 fe  51
7128 : a9 71 85 ff ad dd 71 8d  96
7130 : e3 71 8d 0c 72 8d ff 71  27
7138 : ad de 71 8d e7 71 8d 10  c3
7140 : 72 4c 3c 70 a2 00 a0 06  ae
7148 : 38 bd a2 73 f9 a2 73 dd  b4
7150 : a2 73 f0 09 30 07 38 b9  99
7158 : a2 73 fd a2 73 9d a2 73  1d
7160 : c8 e8 e0 06 d0 e2 a2 00  44
7168 : bd a2 73 fd a3 73 9d a2  a4
7170 : 73 e8 e0 06 d0 f2 a2 00  7f
7178 : 86 02 bd a2 73 c9 fc b0  9e
7180 : 04 c9 04 b0 02 e6 02 e8  b1
7188 : e0 05 d0 ee a5 02 c9 04  96
7190 : b0 03 a9 ff 2c a9 7f 85  45
7198 : 02 a9 00 85 a0 85 a1 85  e7
71a0 : a2 aa 9d a2 73 a0 60 e8  e3
71a8 : d0 f8 60 20 01 71 ee 60  29
71b0 : 70 ee a7 70 a9 f9 85 fe  0e
71b8 : a9 72 85 ff ad e0 71 8d  be
71c0 : 00 73 8d 94 73 a2 03 20  08
71c8 : 3c 70 ce 60 70 ce a7 70  f9
71d0 : a9 86 85 fe a9 73 85 ff  4a
71d8 : a2 00 4c 3c 70 23 10 01  77
71e0 : 23 08 a9 00 85 08 a9 00  b1
71e8 : 85 09 a9 01 85 02 85 3f  79
71f0 : a9 90 85 01 20 93 f3 a9  25
71f8 : 58 85 3a 4c 86 f5 a9 00  ea
7200 : 85 0a a9 e0 85 02 a5 02  14
7208 : 30 fc 60 a9 00 85 08 a9  a3
7210 : 00 85 09 a9 80 85 01 a5  ce
7218 : 01 30 fc a5 01 8d 00 04  aa
7220 : c9 05 f0 03 ee 00 04 60  c8
7228 : a5 1b c9 01 f0 23 c9 02  41
7230 : f0 48 a9 fa 8d 07 1c 8d  ab
7238 : 05 1c e6 1b a9 00 85 0a  2d
7240 : 85 0b a9 b0 85 02 ad 05  f4
7248 : 1c d0 fb 20 00 03 38 b0  2a
7250 : f5 68 68 a2 0a 50 fe b8  78
7258 : ad 01 1c c9 aa f0 04 c9  9c
7260 : 55 d0 f0 e8 d0 ef e6 1b  d5
7268 : a9 00 85 0a 85 0b a9 b0  6d
7270 : 85 02 ad 07 1c 8d 05 1c  bd
7278 : d0 cc 68 68 ad 07 1c 8d  74
7280 : 05 1c a2 0a 50 fe b8 ad  b8
7288 : 01 1c c9 aa f0 04 c9 55  60
7290 : d0 f0 e8 d0 ef ad 05 1c  e5
7298 : a6 35 9d 00 04 20 8f f9  b4
72a0 : a9 3a 8d 07 1c 8d 05 1c  25
72a8 : a9 01 85 02 a2 3b 9a 4c  7a
72b0 : b0 f2 a2 00 86 35 a9 01  3d
72b8 : 85 06 a9 00 85 07 a9 80  e3
72c0 : 85 00 a5 00 30 fc c9 01  c3
72c8 : f0 03 4c a0 ea a9 00 85  68
72d0 : 0a a9 00 85 1b a9 e0 85  ed
72d8 : 02 a5 02 30 fc ee 15 05  d9
72e0 : ee 41 05 ee a6 05 e6 35  27
72e8 : a6 35 e0 06 d0 df a2 00  b8
72f0 : a9 60 9d 00 05 e8 d0 fa  02
72f8 : 60 4c 29 06 20 00 c1 a9  e6
7300 : 00 85 1b 85 0c 20 4b f2  0f
7308 : 85 43 85 35 20 07 d3 4c  59
7310 : da c8 00 00 00 00 00 00  4f
7318 : 00 00 00 00 00 00 00 00  19
7320 : 00 00 20 0e fe 20 00 fe  d9
7328 : a5 35 85 43 a9 09 8d 26  97
7330 : 06 a9 00 8d 28 06 a0 00  f2
7338 : a6 3d a5 39 99 00 03 c8  45
7340 : c8 ad 28 06 99 00 03 c8  e1
7348 : a5 1b 99 00 03 c8 b5 13  55
7350 : 99 00 03 c8 b5 12 99 00  16
7358 : 03 c8 a9 0f 99 00 03 c8  43
7360 : 99 00 03 c8 a9 00 59 fa  c9
7368 : 02 59 fb 02 59 fc 02 59  8e
7370 : fd 02 99 f9 02 ee 28 06  58
7378 : ad 28 06 c5 43 90 bb a9  6f
7380 : 24 85 51 4c 84 fc 20 00  f5
7388 : c1 a2 00 a9 f0 9d 00 03  d2
7390 : e8 d0 fa a9 00 85 06 85  23
7398 : 07 a9 90 85 00 a5 00 30  d6
73a0 : fc 60 00 00 00 00 00 00  cd
```

Listing 2. »PROTECT1«, erster Teil des Schutzprogramms

```
Name : protect2               2710 2ba1
---------------------------------------
2710 : 20 44 e5 a9 0b 8d 20 d0   40
2718 : 78 a2 06 a0 00 cc 12 d0   c7
2720 : d0 fb 8e 21 d0 ad 11 d0   16
2728 : 10 fb a9 0c 8d 21 d0 c8   d8
2730 : ea ea d0 e9 78 a0 fd a2   ca
2738 : 0b cc 12 d0 d0 fb 8e 21   b1
2740 : d0 ea ad 11 d0 10 fb ee   6e
2748 : 21 d0 88 d0 ec a9 0b 8d   71
2750 : 21 d0 20 66 e5 a9 03 85   71
2758 : 9a a9 d1 a0 27 85 22 84   7f
2760 : 23 a9 f2 20 24 ab a9 c6   ec
2768 : a0 28 85 22 84 23 a9 2c   22
2770 : 20 24 ab a0 00 84 c6 a5   2c
2778 : cb c9 40 f0 fa a9 f3 a0   64
2780 : 28 85 22 84 23 a9 28 20   e4
2788 : 24 ab a2 09 86 02 a9 00   6b
2790 : 20 cd bd 20 a5 27 a9 9d   80
2798 : 20 47 ab a6 02 ca e0 ff   15
27a0 : d0 ea 4c c1 27 a9 04 85   0b
27a8 : ff a2 c8 a0 64 a5 cb c9   75
27b0 : 3f f0 0b 88 d0 f7 ca d0   d5
27b8 : f2 c6 ff d0 ec 60 4c 10   4a
27c0 : 27 a9 1b a0 29 85 22 84   e7
27c8 : 23 a9 7d 20 24 ab 4c 1d   2e
27d0 : 2a 11 11 98 20 20 20 20   9e
27d8 : 20 20 20 d5 c3 c3 c3 c3   bc
27e0 : c3 c3 c3 c3 c3 c3 c3 c3   df
27e8 : c3 c3 c3 c3 c3 c3 c3 c3   e7
27f0 : c3 c9 20 20 20 20 20 20   68
27f8 : 20 20 20 20 20 20 20 20   f8
2800 : 20 20 20 c2 20 90 44 49   bb
2808 : 53 4b 20 50 52 4f 54 45   8e
2810 : 43 54 2d 53 59 53 54 45   3f
2818 : 4d 98 20 c2 20 20 20 20   d6
2820 : 20 20 20 20 20 20 20 20   20
2828 : 20 20 20 20 20 ca c3 c3   53
2830 : c3 c3 c3 c3 c3 c3 c3 c3   2f
2838 : c3 c3 c3 c3 c3 c3 c3 c3   37
2840 : c3 c3 c3 cb 20 20 20 20   13
2848 : 20 20 20 20 20 20 20 20   48
2850 : 20 20 b0 c3 c3 c3 c3 c3   16
2858 : c3 c3 c3 c3 c3 c3 c3 c3   57
2860 : c3 c3 c3 c3 c3 c3 c3 c3   5f
2868 : c3 c3 c3 c3 c3 c3 ae 20   cb
2870 : 20 20 20 20 20 20 20 20   70
2878 : 20 20 c2 20 43 4f 50 59   00
2880 : 52 49 47 48 54 20 31 39   cf
2888 : 38 36 20 42 59 20 44 2e   30
2890 : 49 52 49 4f 4e 20 c2 20   70
2898 : 20 20 20 20 20 20 20 20   98
28a0 : 20 20 ad c3 c3 c3 c3 c3   a5
28a8 : c3 c3 c3 c3 c3 c3 c3 c3   a7
28b0 : c3 c3 c3 c3 c3 c3 c3 c3   af
28b8 : c3 c3 c3 c3 c3 c3 bd 20   58
28c0 : 20 20 20 20 20 20 11 11   66
28c8 : 11 98 20 20 20 20 49 4e   f6
28d0 : 53 45 52 54 20 54 41 52   33
28d8 : 47 45 54 20 44 49 53 4b   4d
28e0 : 20 41 4e 44 20 50 52 45   15
28e8 : 53 53 20 41 4e 59 20 4b   dc
28f0 : 45 59 20 0d 11 11 20 20   e6
28f8 : 20 9b 48 49 54 20 52 55   5b
2900 : 4e 2f 53 54 4f 50 20 54   e6
2908 : 4f 20 53 54 4f 50 20 43   45
2910 : 4f 55 4e 54 44 4f 57 4e   e1
2918 : 3a 20 05 11 11 11 98 20   02
2920 : 20 20 20 20 20 20 20 20   20
2928 : 20 20 20 20 20 20 20 12   0c
2930 : ac a2 a2 a2 a2 a2 a2 a2   39
2938 : a2 a2 a2 a2 a2 a2 bb 92   7c
2940 : 20 20 20 20 20 20 20 20   40
2948 : 20 20 20 20 20 20 20 20   48
2950 : 20 20 20 20 20 20 20 20   50
2958 : 20 a1 90 20 50 4c 45 41   70
2960 : 53 45 20 57 41 49 54 20   39
2968 : 98 12 a1 92 20 20 20 20   88
2970 : 20 20 20 20 20 20 20 20   70
2978 : 20 20 20 20 20 20 20 20   78
2980 : 20 20 20 20 20 12 bc 92   67
2988 : a2 a2 a2 a2 a2 a2 a2 a2   87
2990 : a2 a2 a2 a2 a2 12 be 92   5b
2998 : a9 8d 85 fe a9 2a 85 ff   4b
29a0 : a9 00 8d b8 29 20 12 2a   f4
29a8 : a9 4d 20 dd ed a9 2d 20   dd
29b0 : dd ed a9 57 20 dd ed a9   d5
29b8 : 00 20 dd ed a9 05 20 dd   fc
29c0 : ed a9 1f 20 dd ed a0 00   1d
29c8 : b1 fe 20 dd ed c8 c0 1f   22
29d0 : d0 f6 18 98 6d b8 29 90   97
29d8 : 03 ee bd 29 8d b8 29 20   6a
29e0 : fe ed 18 a5 fe 69 1f 90   68
29e8 : 02 e6 ff 85 fe ce 8b 2a   f7
29f0 : d0 b3 20 12 2a a9 4d 20   4a
29f8 : dd ed a9 2d 20 dd ed a9   d8
2a00 : 45 20 dd ed a9 00 20 dd   61
2a08 : ed a9 06 20 dd ed 20 fe   1b
2a10 : ed 60 a9 08 20 0c ed a9   06
2a18 : 6f 20 b9 ed 60 a9 0[illegible] [illegible]5   3e
2a20 : 02 a5 02 8d e0 71 20 ab   98
2a28 : 71 c6 02 d0 f4 20 98 29   9c
2a30 : a9 02 a2 08 a0 61 20 ba   8f
2a38 : ff a2 86 a0 2a a9 05 20   82
2a40 : bd ff 20 c0 ff a9 08 20   cb
2a48 : b1 ff a9 61 20 93 ff a9   81
2a50 : 00 85 fe a9 70 85 ff a9   8e
2a58 : 00 20 dd ed a9 70 20 dd   f8
2a60 : ed a9 03 85 fc a2 00 a1   bb
2a68 : fe 20 dd ed e6 fe d0 f5   41
2a70 : c6 fc f0 05 e6 ff 4c 65   fb
2a78 : 2a a9 08 20 ae ff a9 02   13
2a80 : 20 c3 ff 4c 94 e3 43 2e   dd
2a88 : 4f 42 4a 0a 09 ad 04 1c   13
2a90 : ad 0c 1c 09 0e 8d 0c 1c   21
2a98 : a0 05 b9 00 00 10 2e c9   f6
2aa0 : d0 d0 04 98 4c bd 03 29   fe
2aa8 : 01 f0 07 84 3f a9 0f 4c   8a
2ab0 : 69 f9 aa 85 3d c5 3e f0   4e
2ab8 : 0a 20 7e f9 a5 3d 85 3e   88
2ac0 : 4c 9c f9 a5 20 30 03 0a   31
2ac8 : 10 09 4c 9c f9 88 10 ca   bd
2ad0 : 4c 9c f9 a9 20 85 20 a0   0e
2ad8 : 05 84 3f 20 93 f3 30 1a   c1
2ae0 : c6 3f 10 f7 a4 41 20 95   49
2ae8 : f3 a5 42 85 4a 06 4a a9   40
2af0 : 60 85 20 b1 32 85 22 4c   c2
2af8 : 9c f9 29 01 c5 3d d0 e0   47
2b00 : a5 22 f0 12 38 f1 32 f0   f2
2b08 : 0d 49 ff 85 42 e6 42 a5   1a
2b10 : 3f 85 41 4c 53 03 a2 04   cc
2b18 : b1 32 85 40 dd d6 fe ca   72
2b20 : b0 fa bd d1 fe 85 43 8a   35
2b28 : 0a 0a 0a 0a 0a 85 44 ad   34
2b30 : 00 1c 29 9f 05 44 8d 00   25
2b38 : 1c a6 3d a5 45 c9 40 f0   31
2b40 : 15 c9 60 f0 03 4c bb 03   f8
2b48 : a5 3f 18 69 03 85 31 a9   35
2b50 : 00 85 30 6c 30 00 a9 60   17
2b58 : 85 20 ad 00 1c 29 fc 8d   73
2b60 : 00 1c a9 a4 85 4a a9 01   c1
2b68 : 85 22 4c 69 f9 a4 3f b9   74
2b70 : 00 00 48 10 10 29 78 85   bc
2b78 : 45 98 0a 69 06 85 32 98   40
2b80 : 18 69 03 85 31 a0 00 84   df
2b88 : 30 68 60 a2 5a 20 00 c1   83
2b90 : a9 01 85 0a a9 00 85 0b   23
2b98 : a9 90 85 02 a5 02 30 fc   50
2ba0 : 60 ff ff ff ff ff ff ff   00
```

Listing 3. »PROTECT2«, zweiter Teil des Schutzprogramms

PEEK(2) erfahren Sie nun das Ergebnis des Tests: Enthält diese Adresse den Wert 127, so handelt es sich um ein Original. 255 bedeutet, daß die Diskette kopiert wurde.

Achtung:

1. Beim erstmaligen Starten der Routine wird noch keine Abfrage vorgenommen, sondern lediglich die Vergleichswerte in den Block 16, Track 35 geschrieben. Sie erkennen das an einer Null in der Speicherstelle 2 des C 64.

2. Vor jedem (!) Starten der Routine muß sie neu von der Diskette geladen werden.

3. Ein Punkt, den Sie sich immer vor Augen führen sollten: Durch den Kopierschutz wird Ihre Diskette zwar so gut wie nicht kopierbar, aber nicht »unknackbar«. Das heißt, Sie müssen die Abfrage in Ihrem Programm so gut wie möglich vor fremden Augen schützen, um einen wirkungsvollen Schutz zu erreichen.

4. Die markierten Blöcke auf den Spuren 1 bis 6 und der Block 35/16, der die Zeitwerte enthält, sind aus Sicherheitsgründen für den Schutzmechanismus in der BAM nicht belegt! Das heißt, daß sie aus Versehen durch andere Programme überschrieben werden könnten. Wenn Sie also eine präparierte Diskette mit Programmen füllen möchten, müssen Sie zuerst die Blöcke in der BAM belegen. Dabei hilft Ihnen Listing 4. Wenn die Diskette fertig ist, sollten Sie die Blöcke durch ein Validate wieder freigeben.

Wie baut man nun das Abfragefile in ein Hauptprogramm ein? Eine simple Variante ist es, das File von einem Basic-Lader aus nachladen zu lassen. Im Hauptprogramm sollten dann an verschiedenen Stelle einige Speicherstellen des Abfrageprogramms überprüft werden. Damit verhindern Sie, daß das Programm einfach durch Weglassen der Laderoutine kopiert werden kann.

Eine weitere, wirklich geschickte Methode ist es, die beiden Echtzeituhren der CIAs des C 64 (Adressen 56328 bis 56331 und 56584 bis 56587 im BCD-Format) vom Ladeprogramm aus mit verschiedenen Uhrzeiten zu belegen. Die Differenz merken Sie sich irgendwo im Speicher. Versucht nun jemand, das Ladeprogramm einfach wegzulassen, entfällt logischerweise auch das Stellen dieser Uhren. Im Hauptprogramm können Sie dann darauf entsprechend reagieren. Dies ist wahrscheinlich auch die einzige Möglichkeit, die berühmt-berüchtigten Kopiermodule auszutricksen. Denn diese können den gesamten Speicherinhalt des C 64, nicht jedoch den Inhalt der Portbausteine speichern.

Nun folgen noch einige Informationen zum Programm:

$71DD: Hier befinden sich einige wichtige Informationen über die Abfragedaten.

- Track und Sektor des Blockes, auf den die Abfragedaten gespeichert werden (normal: 35/16)

```
1 PRINT"{CLR}DISKETTE MIT SCHUTZ EINLEGEN
  UND TASTE{2SPACE}DRUECKEN"              <009>
2 POKE 198,0:WAIT 198,1:POKE 198,0        <105>
3 OPEN 1,8,15                             <171>
4 TR=1:SE=0:GOSUB 12                      <052>
5 FOR I=1 TO 6                            <234>
6 TR=I:SE=I:GOSUB 12                      <158>
7 NEXT                                    <017>
8 TR=35:SE=16:GOSUB 12                    <237>
10 CLOSE 1                                <021>
11 :                                      <243>
12 PRINT#1,"B-A 0",TR,SE                  <053>
13 RETURN                                 <071>
```

Listing 4. »BAM NEU«, belegt in der BAM die vom Kopierschutz reservierten Blöcke

- Track, auf dem die erste Abfrage vorgenommen wird
- Tracks, die formatiert und mit einer Abfragemarke versehen werden sollen.

Die Abfrageroutine

Für den Profi soll nun der genaue Ablauf der Abfrageroutine erklärt werden. Diese ist im Programm »protect1« enthalten und steht nach dem Laden des kompletten Systems ab Adresse $7228 im Speicher des C 64. Nach dem Start wird sie zum Diskettenlaufwerk in den RAM-Bereich ab $0500 gesandt. Der Einsprungpunkt liegt bei $72B2 ($058A).

Als erstes wird der Durchlaufszähler in Adresse $35 in der Floppy auf Null gesetzt und Block 0 von Spur 1 der fertig präparierten Diskette gelesen. Er enthält die komplette Routine zum Wechseln einer Spur ohne automatischem Lesen der Sync-Markierung. Sie wird im 1541-Speicher ab Adresse $0300 abgelegt. Diese Routine ist aus dem Markt & Technik-Floppybuch, Seite 334, entnommen.

Sollte beim Einlesen dieses Blocks ein Fehler auftreten, wird in der Floppy sicherheitshalber ein Reset ausgeführt (JMP $EAA0).

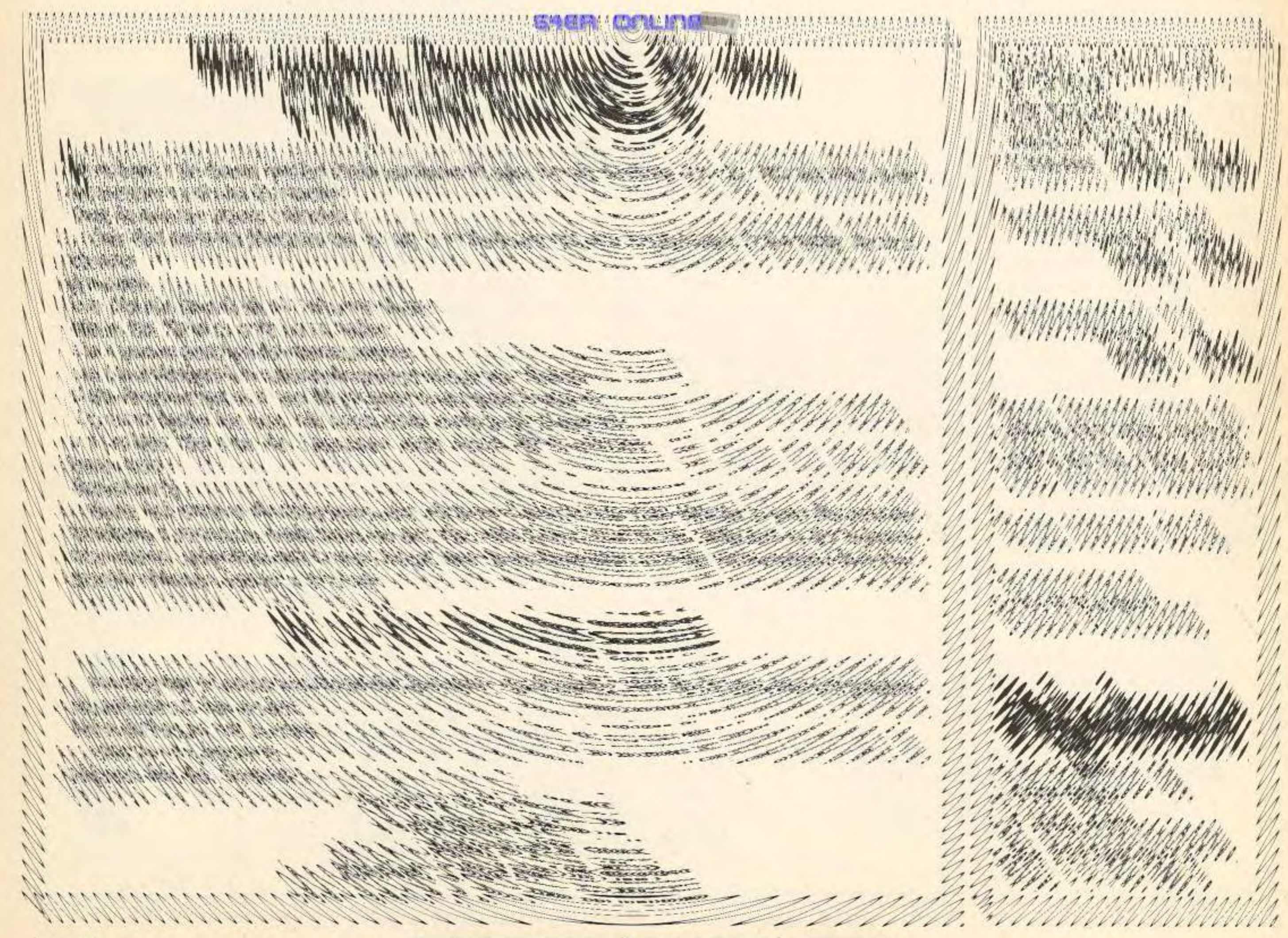
Als erstes wird nun der Schreib-/Lesekopf auf Track 1 gesetzt (hier steht zuerst ein LDA # $00, das aber kurz nach dem Start in ein LDA # $01 geändert wird). Gleichzeitig wird der Laufwerksmotor eingeschaltet und zum Interrupt-Programm ab Adresse $7228 ($0500) gesprungen. Dort angelangt, wird der Inhalt der Adresse $1B geprüft. Beim ersten Durchlauf enthält sie den Wert Null. Das Programm läuft also ohne zu verzweigen weiter. Die nächsten drei Befehle laden den Interrupttimer mit dem neuen Wert $FA (Normalwert: $3A) zur Erhöhung des Wirkungsgrades.

Nun folgt das eigentliche Kernstück der Abfrageroutine: Zuerst werden die Speicherstellen $0A und $0B mit den nötigen Werten für Track und Sektor beschrieben. Danach scheint sich das Programm in einer Endlosschleife zu befinden (SEC:BCS...). Diese Annahme ist falsch, denn der Aufruf der Routine ab Adresse $0300 bewirkt ja ein Wechseln des Tracks.

Ist der Wechsel nach mehrmaligem Durchlauf der »Endlosschleife« erfolgt, wird an den Pufferanfang verzweigt ($0500). Da aber nun die Adresse $1B einen höheren Wert besitzt, verzweigt das Programm zur Adresse $0529 (=$7251 im Speicher des C 64). Dort werden zunächst die Rücksprung-Bytes des »JSR $0300« vom Stapel geholt (PLA:PLA) und die erwähnte Marke auf dem Track gesucht. Dabei muß, wie schon erwähnt, auf die Werte $55 und $AA geprüft werden. Jetzt wird der Trackwert neu gesetzt (in unserem Fall um eins erhöht) und zur Adresse $051E verzweigt (C 64: $727A). Dort beginnt der eigentliche Teil der Zeitmessung: Der Timer wird neu gestellt und die gleiche Prozedur zum Suchen der Marken wiederholt. Ist die Marke gefunden, merkt sich das Programm den aktuellen Timer-Wert ab $0400. Die folgenden Befehle setzen die Interrupt-Zeit auf Normalwert, schalten den Laufwerksmotor aus, setzen den Stackpointer zurück und verlassen schließlich die Interruptroutine.

Wieder bei Adresse $05B5 (C 64: 72DD) angelangt, werden die Werte der anzufahrenden Tracks um eins erhöht. Das Programm wird insgesamt noch sechsmal aufgerufen. Die daraus resultierenden Werte sind danach ab Adresse $0400 zu finden. Zum Schluß löscht sich das Programm durch Überschreiben selbst und endet mit einem RTS.

(Dominik Irion/tr)



32 Funktionstasten

Das endgültige Wort in Sachen »Funktionstastenbelegung« ist gesprochen: Mit dem Programm »key-32« können Sie insgesamt 32 Texte mit je 31 Zeichen über die Funktionstasten abrufen.

Nun werden sich manche sicher fragen, wie man denn mit vier Tasten (<F1>, <F3>, <F5> und <F7>) 32 Funktionen abrufen können soll. Diese enorme Vielzahl wird durch Kombination der vier Funktionstasten des C64 mit den Tasten <SHIFT>, <Commodore> und <CTRL> erreicht.

Zuerst einmal müssen Sie allerdings Listing 1 und Listing 2 abtippen. Beachten Sie hierbei bitte unsere Eingabehinweise auf Seite 92.

Am Anfang wird das Basic-Programm »key-32« geladen und mit RUN gestartet. Das Maschinenprogramm wird automatisch nachgeladen. Nun erscheint die Eingabemaske mit dem Eingabefeld. Die Eingabefolge wird auf dem Bildschirm erklärt. Hier jedoch zusätzlich einige Beispiele.
Zuerst wird eine Zahl von 1 bis 4 eingegeben:
eine 1 für <F1>
eine 2 für <F3>
eine 3 für <F5>
eine 4 für <F7>

Danach eine Zahl von 0 bis 7 für die Sondertasten:
eine 0 für keine Sondertaste
eine 1 für <SHIFT>
eine 2 für <Commodore>
eine 3 für <Commodore + SHIFT>
eine 4 für <CTRL>
eine 5 für <CTRL + SHIFT>
eine 6 für <CTRL + Commodore>
eine 7 für <CTRL + Commodore + SHIFT>

Will man zum Beispiel eine Belegung der Tastenkombination <F3> mit <Commodore> und <SHIFT> erreichen, so gibt man die Zahl 23 ein. Dieses Feld mit den beiden Zahlen wird revers dargestellt. Eine eventuelle Falscheingabe wird angezeigt und verhindert.

Nun gibt es zwei Möglichkeiten, um fortzufahren. Entweder gibt man direkt nach den zwei Zahlen einen Klammeraffen (@) ein, oder man tippt für diese Tastenkombination den Belegtext mit einem abschließenden Klammeraffen ein. Ersteres listet die Tastenbelegung (man kann also sehen, ob die Taste belegt ist). Die zweite Möglichkeit speichert den eingegebenen Text mit der definierten Tastenkombination.

Beispiel: Belegung der <F1>-Taste ohne Sondertaste.
Belegen: 10load"$",8@
Listen : 10@

Mit der Cursor-Taste bewegt man sich im Eingabefeld. Die <←>-Taste löscht die angezeigte Eingabe (nicht aber die schon gespeicherte Tastenbelegung) und setzt den Cursor an den Textbeginn.

Will man die gesamte Tastenbelegung betrachten, drückt man <£>. Dies zeigt aus Platzgründen nur die ersten 16 Tasten. Drückt man nun <2>, so kann man das zweite Blatt mit den Tasten 17 bis 32 betrachten. Das Tippen der Taste <H> führt uns wieder in das Hauptprogramm. Will man die fertige Tastenbelegung auf Diskette speichern, so verwendet man die Tasten <SHIFT+£>. Dadurch wird das eigentliche Anwenderprogramm (m-key-32) erzeugt. Der Druck auf die Tasten <CTRL> und <←> ruft dieses Programm auf. Die neue Bildschirmanzeige und der Piepston bei Tastendruck geben die Funktionsbereitschaft an.

Für den Gebrauch (Tasten wurden schon belegt)
Das Maschinenprogramm wie folgt laden:
LOAD"M-KEY-32",8,1
Danach den Befehl NEW <RETURN> eingeben und das Programm durch SYS 52000 starten.

Programmhinweise
Die Programmfunktionen werden durch die REM-Anweisungen in den Listings ausgiebig beschrieben. Die Variablen haben folgende Bedeutung:

A$,B$ = Ein- und Ausgabevariablen
SP = Cursorspalte
CR = Zeiger, der die zur Eingabe anstehende Bildschirmposition anzeigt
FR = Funktionstastenrubrik (-spalte)
SR = Sondertastenrubrik (-spalte)
I = Schleifenzähler
T1,T2 = Die ersten zwei Werte im Eingabefeld (F+S)
Kf = Klammeraffenabfrage im dritten Eingabefeld
FZ,SZ = Funktionstastenzahl (-wert) Sondertasten (-zahl)
PB = POKE und PEEK Basisadresse des Tastentextes
PE = POKE und PEEK Endadresse des Tastentextes
ZE = Zeichen in dieser Adresse
FL = Flag für Textende
SA = Startadresse (-position) des Ausgabetextes
LO = Schleife für 31 einzelne Zeichen des Ausgabetextes
HI = Schleife für 16 verschiedene Tasten
PS = POKEstelle (-position)
PW = PEEK- und POKEwert

Soviel zu dem Basic-Programm key-32.

Das Maschinenprogramm m-key-32 belegt den Speicher von $CB20 bis CFFF. Im Bereich $CC00 bis $CFFF sind die Inhalte der einzelnen Tasten abgelegt.

(Siegbert Werner/tr)

```
10 REM       --- K E Y   3 2 ---                <003>
20 REM COPYRIGHT: SIEGBERT WERNER               <192>
30 REM BEETHOVENSTR. 59 SIEGEN 31               <022>
40 POKE 53280,0:POKE 53281,0:POKE 646,15        <066>
42 A=A+1                                        <121>
44 IF A<2 THEN LOAD"M-KEY-32",8,1               <147>
50 SP=3:REM-------CURSOR IN SPALTE 3            <151>
60 GOSUB 580:REM BILDSCHIRMAUFBAU               <180>
70 GOTO 190:REM----CURSOR SETZEN                <020>
78 REM------------TASTATURABFRAGE               <127>
80 POKE 198,0:WAIT 198,1                        <052>
90 GET A$                                       <044>
97 REM------------UNERWUENSCHTE TASTE?          <000>
98 REM------------Z.B.:RETURN,HOME,ECT.         <044>
99 REM------------DANN NICHT REAGIEREN          <133>
100 IF ASC(A$)=17 OR ASC(A$)=145 OR ASC(A$
    )=148 OR ASC(A$)=20 OR ASC(A$)=147 OR
    ASC(A$)=19 THEN 190                         <239>
105 IF ASC(A$)=13 THEN 190                      <119>
107 REM------------ABFRAGE DER ERLAUBTEN        <064>
108 REM------------TASTEN                       <065>
109 REM------------CURSOR LINKS-TASTE?          <021>
```

Listing 1. »key-32«, das Basic-Programm zum Belegen der Funktionstasten

```
110 IF ASC(A$)=157 THEN SP=SP-2                  <061>
120 IF SP<2 THEN SP=2:REM-CURSORFELDANFANG       <235>
129 REM-------------CTRL MIT ← TASTE?            <142>
130 IF ASC(A$)=6 THEN GOTO 340  :REM PRG-S
    TART!                                        <056>
135 IF A$="£"THEN 880                            <034>
137 IF A$="£"THEN 1810                           <196>
140 IF A$="←"THEN GOTO 290                       <084>
150 PRINT A$                                     <072>
160 IF A$="@"THEN GOSUB 400                      <105>
170 IF SP=36 THEN 190:REM-CURSORFELDENDE         <067>
180 SP=SP+1:REM---CURSOR EINS RECHTS             <024>
189 REM-------------CURSOR SETZEN                <170>
190 POKE 211,SP                                  <234>
200 POKE 214,23                                  <167>
210 SYS 58732                                    <067>
218 REM-------------SETZEN DES ZEIGERS           <131>
219 REM-------------AUF DAS EINGABEFELD          <111>
220 CR=1024+(PEEK(214)+1)*40+PEEK(211)           <053>
230 POKE CR,30:POKE CR-1,67:POKE CR+1,67         <157>
238 REM-------------INVERTIEREN DER              <193>
239 REM-------------ZAHLEN FUER F+STASTE         <036>
240 FR=PEEK(1947):SR=PEEK(1948)                  <179>
250 IF FR<=127 THEN FR=FR+128                    <115>
260 IF SR<=127 THEN SR=SR+128                    <146>
270 POKE 1947,FR:POKE 1948,SR                    <255>
279 REM-------------NEUE TASTE ABFRAGEN          <100>
280 GOTO 80                                      <004>
289 REM-------------EINGABEFELD LOESCHEN         <071>
290 POKE CR,67:POKE CR-1,67:POKE CR+1,67         <031>
300 SP=3                                         <142>
310 A$=""                                        <085>
320 FOR I=0 TO 33:POKE 1947+I,32:NEXT I          <145>
330 GOTO 190                                     <148>
339 REM-------------MASCHINENPRG.-START!         <092>
340 :A$=" "                                      <121>
350 SYS 52000:REM-IRQ AUF NEUE ROUTINE!          <135>
359 REM-------------BEREITSCHAFTSBILD            <020>
360 PRINT"{CLR}"                                 <094>
370 PRINT"{14SPACE}K E Y{3SPACE}3 2{13SPAC
    E}"                                          <172>
380 PRINT"-----------------------------------
    ---------";                                  <106>
382 POKE 211,0:POKE 214,23:SYS 58732             <227>
384 PRINT"-----------------------------------
    ---------";                                  <110>
388 POKE 211,0:POKE 214,2:SYS 58732              <022>
390 END                                          <138>
398 REM-------------FUER F+S NUR ZAHLEN          <023>
399 REM-------------ERLAUBEN!                    <032>
400 T1=PEEK(1947)-128:T2=PEEK(1948)-128          <067>
402 IF T1<49 OR T1>52 THEN 840                   <113>
404 IF T2<48 OR T2>55 THEN 840:REM-FEHLER        <249>
408 REM-------------SPEICHERN? - ZEIGEN?         <007>
410 KF=PEEK(1949)                                <238>
420 IF KF<>0 THEN 510                            <226>
429 REM-------------KEYBELEGUNG ZEIGEN           <045>
430 FZ=PEEK(1947):SZ=PEEK(1948)                  <181>
440 PB=51968+(FZ-176)*256+(SZ-176)*32            <103>
450 FOR I=0 TO 31:PE=PB+I:ZE=PEEK(PE)            <205>
460 IF ZE=0 THEN 490:REM-TEXTENDE=@=0            <097>
470 POKE 1949+I,ZE:REM-ZEICHENAUSGABE            <068>
480 NEXT I                                       <054>
490 SP=4:RETURN                                  <065>
500 REM-------------KEYBELEG. SPEICHERN          <213>
510 FZ=PEEK(1947):SZ=PEEK(1948)                  <005>
520 PB=51968+(FZ-176)*256+(SZ-176)*32            <185>
530 FOR I=0 TO 31:PE=1949+I:ZE=PEEK(PE)          <038>
540 POKE PB+I,ZE                                 <197>
550 IF ZE=0 THEN 290                             <023>
560 NEXT I                                       <136>
570 GOTO 290                                     <142>
579 REM-------------BILDSCHIRMAUFBAU             <008>
580 PRINT"{CLR,RVSON,40SPACE}";                  <095>
590 PRINT"{RVSON,14SPACE}K E Y{3SPACE}3 2{
    15SPACE}";                                   <072>
600 PRINT"{RVSON,40SPACE,RVOFF}";                <107>
610 PRINT                                        <204>
620 PRINT" UCCCCCCCCI UCCCCCCCCCCCCCCCCCCC
    CCCCCCI ";                                   <188>
630 PRINT" BF-TASTENB BS-TASTEN{17SPACE}B
    ";                                           <062>
640 PRINT" B={7SPACE}B B={24SPACE}B ";           <002>
650 PRINT" B1= F1+F2B B0= OHNE SONDERTASTE
    {6SPACE}B ";                                 <138>
660 PRINT" B2= F3+F4B B1= SHIFT{17SPACE}B
    ";                                           <160>
670 PRINT" B3= F5+F6B B2= COMMODORE{13SPAC
    E}B ";                                       <067>
680 PRINT" B4= F7+F8B B3= SHIFT+COMMODORE{
    7SPACE}B ";                                  <197>
690 PRINT" JCRCCCCCCK B4= CTRL{18SPACE}B "
    ;                                            <187>
700 PRINT"{3SPACE}B{8SPACE}B5= SHIFT+CTRL{
    12SPACE}B ";                                 <061>
710 PRINT"{3SPACE}B{8SPACE}B6= COMMODORE+C
    TRL{8SPACE}B ";                              <220>
720 PRINT"{3SPACE}B↑CCCCCCCW7= SHIFT+COMMO
    DORE+CTRL{2SPACE}B ";                        <139>
730 PRINT"{3SPACE}FB{7SPACE}JCCCCCCCCCCCCC
    CCCCCCCCCCCCK ";                             <147>
740 PRINT"{3SPACE}BS{7SPACE}UCCCCCCCCCCCCC
    CCCCCCCCCCCCI ";                             <056>
750 PRINT"{3SPACE}BB↑CCCCCCWTEXTEINGABE MI
    T @ BEENDENB ";                              <246>
760 PRINT"{3SPACE}BBB{6SPACE}BNUR @ HINTER
     F+S= AUSGABEB ";                            <175>
770 PRINT"{3SPACE}BBB{6SPACE}BCTRL/← =STAR
    T{2SPACE}£= KEYLISTB ";                      <146>
780 PRINT"{3SPACE}BBB{6SPACE}BSHIFT/£=SAVE
    N{2SPACE}←= LOESCH.B ";                      <088>
785 PRINT"{3SPACE}BBB{6SPACE}JCCCCCCCCCCCC
    CCCCCCCCCCCCCK ";                            <214>
790 PRINT" UC+++CCCCCCCCCCCCCCCCCCCCCCCCCC
    CCCCCCI"                                     <049>
800 PRINT" C{36SPACE}C"                          <092>
810 PRINT" JCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
    CCCCCCK";                                    <071>
820 RETURN                                       <116>
830 REM--------------EINGABEFEHLER               <076>
831 REM--------------KENNTLICH MACHEN!           <151>
840 FOR I=0 TO 31:READ Q:POKE 1949+I,Q:NEX
    T                                            <249>
845 POKE CR,67:POKE CR-1,67:POKE CR+1,67         <078>
850 RESTORE:SP=3:GOTO 190                        <188>
860 DATA 14,21,18,32,6,5,19,20,7,5,12,5,7,
    20,5,32,26,1,8,12,5,14                       <179>
870 DATA 32,9,14,32,6,43,19,33,32,32             <208>
879 REM--------------KEYLISTING BLATT1           <109>
880 GOSUB 900:SA=52224:GOSUB 1600                <158>
890 GOTO 1360                                    <052>
899 REM--------------KEYLISTING BILD             <083>
900 PRINT"{CLR,RVSON,40SPACE}";                  <161>
910 PRINT"{RVSON,2SPACE}K E Y{3SPACE}3 2{3
    SPACE}-BLATT 1-{2SPACE}WERNER-MADE{2SP
    ACE}";                                       <232>
920 PRINT"{RVSON,40SPACE,RVOFF}";                <173>
930 PRINT                                        <014>
940 PRINT"{4SPACE}↑CCCSHIFT{5SPACE}1 / 2=S
    EITENWAHL"                                   <134>
950 PRINT"{4SPACE}B↑CCCOMMODORE H{4SPACE}=
    RUECKSPRUNG"                                 <069>
960 PRINT"{4SPACE}BB↑CCTRL{6SPACE}CTRL/←=S
    TART HAUPTPRG.";                             <007>
970 PRINT"{4SPACE}BBB"                           <140>
1000 PRINT" F1+WWW=.......................
     ........"                                   <222>
1010 PRINT" F1+QWW=.......................
     ........"                                   <229>
1020 PRINT" F1+WQW=.......................
     ........"                                   <236>
1030 PRINT" F1+QQW=.......................
     ........"                                   <245>
1040 PRINT" F1+WWQ=.......................
     ........"                                   <252>
1050 PRINT" F1+QWQ=.......................
     ........"                                   <003>
1060 PRINT" F1+WQQ=.......................
     ........"                                   <010>
1070 PRINT" F1+QQQ=.......................
     ........"                                   <017>
1080 PRINT" F2+WWW=.......................
     ........"                                   <080>
1090 PRINT" F2+QWW=.......................
```

```
         ........."                                 <087>
1100 PRINT" F2+WOW=.........................
         ........."                                 <094>
1110 PRINT" F2+OOW=.........................
         ........."                                 <101>
1120 PRINT" F2+WWO=.........................
         ........."                                 <108>
1130 PRINT" F2+OWO=.........................
         ........."                                 <115>
1140 PRINT" F2+WOO=.........................
         ........."                                 <122>
1150 PRINT" F2+OOO=.........................
         .........";                                <090>
1152 RETURN                                         <194>
1350 REM-------------WELCHE AUSWAHLTASTE?           <144>
1360 POKE 198,0:WAIT 198,1                          <062>
1370 GET B$                                         <062>
1380 IF ASC(B$)=6 THEN 340                          <215>
1390 IF B$="H"THEN 50                               <194>
1400 IF B$="1"THEN 880                              <143>
1405 IF B$="2"THEN 1500                             <231>
1410 GOTO 1360                                      <064>
1498 REM-------------KEYLISTING BLATT2              <249>
1499 REM-------------AENDERUNG D. BILDES            <068>
1500 POKE 1087,178                                  <057>
1502 FOR I=0 TO 7:POKE 1346+I*40,53:NEXT I          <080>
1504 FOR I=0 TO 7:POKE 1666+I*40,55:NEXT I          <202>
1507 REM-------------LESEADRESSE AENDERN            <074>
1508 SA=52736:GOSUB 1600                            <045>
1510 GOTO 1360                                      <164>
1599 REM-------------AUSGABESCHLEIFE                <254>
1600 FL=1:FOR HI=0 TO 15                            <072>
1610 FOR LO=0 TO 30                                 <249>
1620 PW=PEEK(SA+HI*32+LO)                           <049>
1630 PS=1352+HI*40+LO                               <137>
1640 IF PW=0 THEN FL=0                              <247>
1650 IF FL=0 THEN PW=46:REM PUNKTE NACH TE
     XT                                             <207>
1660 POKE PS,PW                                     <225>
1670 NEXT LO                                        <242>
1680 FL=1                                           <104>
1690 NEXT HI                                        <134>
1700 RETURN                                         <234>
1800 REM              SAVEN DER BELEGUNG            <076>
1810 OPEN 1,8,1,"@:M-KEY-32"                        <244>
1820 A$=CHR$(32):B$=CHR$(203)                       <216>
1830 PRINT#1,A$;B$;                                 <037>
1840 FOR I=52000 TO 53248:A$=CHR$(PEEK(I))
     :PRINT#1,A$;:NEXT I                            <042>
1950 CLOSE 1                                        <183>
1960 GOTO 50:REM------ZURUECK ZUM PRG.              <047>
```

Listing 1. »key-32«, das Basic-Programm zum Belegen der Funktionstasten (Schluß)

```
Name : m-key-32                  cb20 d001
------------------------------------------
cb20 : a9 00 8d 20 d0 8d 21 d0   d0
cb28 : 20 44 e5 a9 04 85 ff a9   d9
cb30 : 0e 85 fe a9 3c 85 fc a9   2d
cb38 : cb 85 fd d0 09 2d 0b 05   8f
cb40 : 19 20 20 33 32 2d a0 09   f9
cb48 : b1 fc 91 fe 88 d0 f9 a9   06
cb50 : ff 8d 06 d4 8d 18 d4 a9   72
cb58 : 00 8d 05 d4 a9 0d 8d 01   36
cb60 : d4 a9 0a 8d 00 d4 78 a9   19
cb68 : 73 8d 14 03 a9 cb 8d 15   61
cb70 : 03 58 60 a5 cb c9 40 d0   1a
cb78 : 07 a9 00 8d 04 d4 f0 1b   e6
cb80 : 48 a9 21 8d 04 d4 68 c9   b3
cb88 : 5c f0 10 c9 03 f0 0f c9   21
cb90 : 04 f0 0b c9 05 f0 07 c9   90
cb98 : 06 f0 03 4c 31 ea aa bd   f1
cba0 : ed cb 85 ff ad 8d 02 aa   79
cba8 : bd f4 cb 85 fe a5 d6 aa   51
cbb0 : 20 ff e9 a9 00 85 d3 20   3b
cbb8 : 6c e5 a5 d6 85 28 a9 28   ec
cbc0 : 85 71 a9 00 85 29 85 72   05
cbc8 : 20 57 b3 86 fc 84 fd a9   91
cbd0 : 04 18 65 fd 85 fd a0 00   c4
cbd8 : b1 fe c9 00 f0 09 91 fc   12
cbe0 : c8 c0 20 f0 02 d0 f1 98   ce
cbe8 : 85 d3 20 6c e5 d0 ac ea   5a
cbf0 : cf cc cd ce 00 20 40 60   35
cbf8 : 80 a0 c0 e0 00 8b 85 99   ba
cc00 : 0c 0f 01 04 22 24 22 2c   79
cc08 : 38 00 8d 02 aa bd f4 cb   e8
cc10 : 85 fe a5 d6 aa 20 ff e9   d8
cc18 : a9 00 85 d3 20 6c e5 a5   e5
cc20 : 0c 0f 01 04 22 3a 2a 22   56
cc28 : 2c 38 2c 31 00 20 57 b3   67
cc30 : 86 fc 84 fd a9 04 18 65   fb
cc38 : fd 85 fd a0 00 10 f9 68   c4
cc40 : 0c 4f 22 20 20 20 20 20   44
cc48 : 2a 22 2c 38 2c 31 00 ad   3d
cc50 : bc 02 a0 00 91 fb a9 14   fd
cc58 : 8d 16 03 a9 c2 8d 17 03   e1
cc60 : 0c 4f 22 00 3a 00 51 c3   0d
cc68 : ad 11 d0 09 10 8d 11 d0   46
cc70 : 60 8d ab 02 08 68 29 ef   0a
cc78 : 8d aa 02 8e ac 02 8c ad   15
cc80 : 05 13 20 06 15 0e 0b 14   ee
cc88 : 09 0f 0e 09 05 12 14 20   2f
cc90 : 04 0f 03 08 20 10 12 09   bb
cc98 : 0d 01 20 0f 04 05 12 00   c1
cca0 : 0c 4f 22 20 20 20 20 20   a4
cca8 : 20 20 20 20 20 20 20 20   a8
ccb0 : 20 20 20 22 2c 38 2c 31   c4
ccb8 : 00 29 10 f0 08 20 65 cc   20
ccc0 : 0c 0f 01 04 22 00 bc 02   2e
ccc8 : 50 1f 38 ad a9 02 ed bd   49
ccd0 : 02 8d b1 02 ad a8 02 ed   49
ccd8 : be 02 0d b1 02 d0 67 ad   b0
cce0 : 3f 22 36 34 27 05 12 22   6c
cce8 : 00 30 12 4e bc 02 [illegible] [illegible]   08
ccf0 : ae ae 02 9a a9 cc 48 a9   3f
ccf8 : 70 48 4c ba cd 20 65 cc   04
cd00 : 08 01 0c 0c 0f 20 36 34   41
cd08 : 27 05 12 20 06 12 05 01   42
cd10 : 0b 13 21 00 20 08 09 10   74
cd18 : 10 05 0c 00 39 30 31 32   ec
cd20 : 0c 09 13 14 00 02 85 fb   16
cd28 : 86 fc 20 49 c3 20 cb c4   54
cd30 : 20 c7 c5 20 e4 ff f0 fb   b3
cd38 : c9 4a d0 0a a9 01 8d bc   ee
cd40 : 0f 10 05 0e 31 2c 34 3a   14
cd48 : 03 0d 04 31 3a 0c 09 13   47
cd50 : 14 00 a9 40 d0 0a 20 f2   9a
cd58 : cd 08 68 8d aa 02 a9 80   57
cd60 : 06 09 0e 04 00 0c 00 20   90
cd68 : 49 c2 20 65 cc ad bc 02   f8
cd70 : f0 37 a2 00 ad 11 d0 a8   9c
cd78 : 29 10 f0 10 98 29 ef 8d   95
cd80 : 04 15 0d 10 00 0c ca d0   81
cd88 : fd 88 d0 fa 78 a9 47 8d   6a
cd90 : 04 dc 8e 05 dc ad 0e dc   74
cd98 : 29 80 09 11 8d 0e dc a9   76
cda0 : 01 15 14 0f 31 30 2c 31   bb
cda8 : 30 00 ae 02 0a 78 ad bb   57
cdb0 : 02 ae ba 02 8d 14 03 8e   9b
cdb8 : 15 03 ad a8 02 48 ad a9   3c
cdc0 : 12 05 0e 15 0d 02 05 12   94
cdc8 : 30 2c 31 00 ac ad 02 40   1b
cdd0 : 20 8d c2 8d be 02 20 8d   b1
cdd8 : c2 8d bd 02 20 8d c2 8d   a5
cde0 : 12 05 0e 15 0d 02 05 12   b4
cde8 : 31 30 30 2c 31 30 00 15   82
cdf0 : 03 60 ad 14 03 ae 15 03   11
cdf8 : 8d b8 02 8e b9 0b 05 19   6e
ce00 : 13 19 13 34 39 31 35 32   42
ce08 : 20 3a 12 05 0d 20 24 03   d3
ce10 : 30 30 30 00 ab 10 03 a2   f1
ce18 : 29 2c a2 28 20 40 c3 20   58
ce20 : 13 19 13 33 36 38 36 34   52
ce28 : 20 3a 12 05 0d 20 24 39   5f
ce30 : 30 30 30 00 86 02 91 f3   2b
ce38 : 68 c8 e8 e0 08 d0 e9 20   49
ce40 : 13 19 13 38 2a 31 36 1e   ed
ce48 : 33 20 3a 12 05 0d 20 24   de
ce50 : 38 30 30 30 00 20 7e c2   33
ce58 : 20 b8 c2 a2 08 a9 00 85   b2
ce60 : 3f 10 05 05 0b 28 34 33   b3
ce68 : 29 2b 10 05 05 0b 28 34   7d
ce70 : 34 29 2a 32 35 36 00 aa   64
ce78 : 81 fb c1 fb d0 ee 20 67   ba
ce80 : 3f 10 05 05 0b 28 34 35   d7
ce88 : 29 2b 10 05 05 0b 28 34   9d
ce90 : 36 29 2a 32 35 36 00 a9   84
ce98 : 21 20 d2 ff 20 23 c3 a0   e9
cea0 : 3f 22 03 0f 10 19 12 09   b7
cea8 : 07 08 14 3a 20 13 09 05   c9
ceb0 : 07 02 05 12 14 20 17 05   e5
ceb8 : 12 0e 05 12 22 00 c1 fb   76
cec0 : 13 19 13 36 34 37 33 38   26
cec8 : 20 20 3a 12 05 0d 20 20   43
ced0 : 12 05 13 05 14 00 3c cf   9c
ced8 : a9 14 85 fb a9 02 05 ff   2b
cee0 : 10 4f 35 33 32 38 30 2c   4a
cee8 : 30 3a 10 4f 35 33 32 38   4a
cef0 : 31 2c 30 3a 10 4f 36 34   47
cef8 : 36 2c 37 00 f2 cf 85 fb   ce
cf00 : 0f 50 31 2c 38 2c 31 35   1d
cf08 : 2c 22 13 3a 20 20 20 20   15
cf10 : 20 20 20 20 20 20 20 22   14
cf18 : 3a 03 0c 4f 31 00 fb 88   d5
cf20 : 0f 50 31 2c 38 2c 31 35   3d
cf28 : 2c 22 0e 3a 20 20 20 20   f4
cf30 : 20 20 20 20 20 20 20 20   30
cf38 : 22 3a 03 0c 4f 31 00 a9   8c
cf40 : 10 4f 32 31 31 2c 20 20   e0
cf48 : 20 3a 10 4f 32 31 34 2c   49
cf50 : 20 20 20 3a 13 19 13 35   80
cf58 : 38 37 33 32 00 c7 c9 20   e5
cf60 : 13 19 13 31 36 1e 33 2a   60
cf68 : 00 fe 78 a9 03 85 01 a2   e0
cf70 : 10 b1 fd 91 fb c8 d0 f9   48
cf78 : e6 fc e6 fe ca d0 f2 a9   c8
cf80 : 13 19 13 00 60 48 c9 4a   e9
cf88 : d0 10 a0 27 b9 00 02 91   34
cf90 : d1 88 10 f8 68 c6 d6 4c   79
cf98 : d6 c2 a0 06 d9 d7 cf d0   f5
cfa0 : 04 01 13 20 17 01 12 20   f0
cfa8 : 04 09 05 20 06 01 0c 13   35
cfb0 : 03 08 05 20 14 01 13 14   ba
cfb8 : 05 00 d0 f8 4c d1 c2 20   af
cfc0 : 0c 0f 01 04 22 24 22 2c   39
cfc8 : 38 00 0c 49 00 dc cf 48   e3
cfd0 : 60 28 29 21 45 59 51 48   a8
cfd8 : 5a 4e 55 44 4b 4d 52 ce   3d
cfe0 : 05 0e 04 05 20 05 0e 04   f9
cfe8 : 05 20 05 0e 04 05 20 05   f3
cff0 : 0e 04 05 20 05 0e 04 05   21
cff8 : 20 05 0e 04 05 20 00 ff   f0
d000 : 00 00 00 00 00 00 00 00   01
```

Listing 2. »m-key-32«, der Maschinenspracheteil von »key-32«

Acht kleine Hilfsprogramme

Kurze, leistungsstarke Maschinensprache-Routinen sind das Lebensbalsam eines jeden Programmierers. Wir stellen Ihnen acht Miniprogramme zum Lesen von Speicherstellen unter dem ROM, zum schnellen Durchsuchen von Variablenfeldern und vielem mehr vor.

Bitte beachten Sie beim Eintippen unserer Listings unsere Eingabehinweise auf Seite 92. Die vorgestellten Programme können Sie entweder einzeln abtippen und bei Bedarf laden oder aber sich aus allen Routinen ein kleines Utility-Programm zusammenstellen:

Tippen Sie die Listings 1 bis 8 ab und speichern sie. Dann laden Sie die Programme nacheinander (den Zusatz »,8,1« nicht vergessen), geben jedesmal NEW ein und speichern schließlich mit einem Maschinensprache-Monitor den Speicherbereich von $C000 bis $C6AE zum Beispiel unter dem Namen »Toolkit« auf Diskette. Bei Bedarf laden Sie dann diese Zusammenstellung aller Programme, die Sie wie nachfolgend bedienen. Unsere Programmservice-Diskette zu diesem Sonderheft enthält dieses File bereits.

Pause (Listing 1)
Format: SYS 49152,Sekundenzahl*60

Das laufende Programm wird für eine definierte Zeitspanne angehalten.
SYS 49152,102 entspricht einer Pause von 1,75 Sekunden.

```
Name : pause                    c000 c02f
-----------------------------------------
c000 : 20 fd ae 20 8a ad 20 f7   55
c008 : b7 a2 00 86 a1 86 a2 a6   07
c010 : a2 e4 14 d0 fa a6 a1 e4   79
c018 : 15 d0 f4 60 fd ff 00 02   c2
c020 : fd ff 00 02 fd ff 00 02   41
c028 : fd ff 00 02 fd ff 00 2c   9d
```

Listing 1. »pause«, definierte Pause für Basic-Programme

Underromread (Listing 2)
Format: SYS 49200,Adresse

Dieses Programm dient zum Auslesen von Speicherstellen »unter« dem Basic-Interpreter oder dem Betriebssystem. In Basic können diese nämlich nicht ausgelesen werden, weil dazu der Interpreter abgeschaltet werden müßte.
SYS 49200,60000 liest Speicherstelle 60000 unter dem Betriebssystem aus und übergibt den Wert an Adresse 2.

```
Name : underromread             c030 c057
-----------------------------------------
c030 : 20 fd ae 20 8a ad 20 f7   85
c038 : b7 a6 14 86 f7 a6 15 86   2e
c040 : f8 78 a2 35 86 01 a0 00   b7
c048 : b1 f7 85 02 a2 37 86 01   96
c050 : 60 ff 00 02 fd ff 00 05   da
```

Listing 2. »underromread« liest Speicherinhalte unterm ROM

Arrayseek (Listing 3)
Format: SYS 49240,Arrayname,gesuchter Text

Ein eindimensionales Stringarray wird nach einem bestimmten String durchsucht. Die Stringnummer steht nachher in 176/177. Wurde der String nicht gefunden, erscheint hier 0.
SYS 49240,MI,"64'ER" durchsucht das Array MI$ nach dem String »64'ER«.

```
Name : arrayseek                c058 c15c
-----------------------------------------
c058 : 20 22 c1 20 57 e2 a6 22   69
c060 : 86 a6 a6 23 86 a7 85 a5   4e
c068 : 20 be c0 a5 9f 85 fa a5   29
c070 : 9e 18 69 07 85 f9 d0 02   c5
c078 : e6 fa a2 01 86 fb ca 86   25
c080 : fc 20 06 c1 85 a8 86 a9   51
c088 : 84 aa a4 a5 c4 a8 d0 15   3e
c090 : 88 b1 a6 d1 a9 d0 0e 88   3f
c098 : c0 ff d0 f5 a6 fb 86 b0   10
c0a0 : a6 fc 86 b1 60 e6 fb d0   6b
c0a8 : 02 e6 fc a6 fb e4 fd d0   b2
c0b0 : d0 a6 fc e4 fe d0 ca a2   96
c0b8 : 00 86 b0 86 b1 60 a5 2f   0b
c0c0 : 85 9e a5 30 85 9f a5 31   52
c0c8 : 85 f9 a5 32 85 fa e6 f9   b9
c0d0 : a0 00 b1 9e d9 ea 07 d0   63
c0d8 : 16 c8 b1 9e 29 7f d9 ea   5e
c0e0 : 07 d0 0c a0 06 b1 9e 85   da
c0e8 : fd 88 b1 9e 85 fe 60 e6   09
c0f0 : 9e d0 02 e6 9f a6 9e e4   c7
c0f8 : f9 d0 d5 a6 9f e4 fa d0   52
c100 : cf a2 64 20 37 a4 a5 f9   61
c108 : 18 69 03 85 f9 90 02 e6   40
c110 : fa a0 00 b1 f9 48 c8 b1   f9
c118 : f9 48 c8 b1 f9 a8 68 aa   79
c120 : 68 60 a0 00 8c ea 07 8c   36
c128 : eb 07 a0 00 20 fd ae 20   ac
c130 : 79 00 85 02 20 73 00 a5   34
c138 : 02 c9 2c f0 18 c9 41 90   3e
c140 : 0f c9 5b b0 0b c8 c0 03   21
c148 : b0 06 99 e9 07 4c 2f c1   b2
c150 : a2 64 4c 37 a4 60 a2 64   bf
c158 : 4c 37 a4 60 00 a9 0c 86   00
```

Listing 3. »arrayseek« durchsucht ein Variablenfeld nach Text

Screen-Tool (Listing 4)
Screen-Tool umfaßt einige Routinen, die das Arbeiten mit dem Bildschirm etwas erleichtern.

1) Cursor setzen:
   SYS 49500,Zeilen-Nr,Spalten-Nr
2) Zeilen löschen
   SYS 49503,erste Zeile,letzte Zeile
3) Bildschirmteil invertieren/reinvertieren
   SYS 49506,Zeilen-Nr,Spalten-Nr,Anzahl Zeichen
   SYS 49521,Zeilen-Nr,Spalten-Nr,Anzahl Zeichen

Entgegen der normalen Betriebssystempraxis beginnt Screen-Tool bei der Numerierung der Zeilen beziehungsweise Spalten nicht mit 0, sondern mit 1.

```
Name : screen-tool              c15c c224
-----------------------------------------
c15c : 4c c6 c1 4c dc c1 20 80   63
c164 : c1 b1 f7 09 80 91 f7 88   a2
c16c : c0 ff d0 f5 60 20 80 c1   ab
c174 : b1 f7 29 7f 91 f7 88 c0   d8
c17c : ff d0 f5 60 20 fd ae 20   5a
c184 : 9e b7 8e b2 02 20 fd ae   6e
c18c : 20 9e b7 8e b3 02 ce b2   a7
c194 : 02 ce b3 02 a9 00 85 f7   cb
c19c : a9 04 85 f8 a5 f7 18 69   15
c1a4 : 28 85 f7 90 02 e6 f8 ce   78
c1ac : b2 02 d0 f0 a5 f7 18 6d   07
c1b4 : b3 02 85 f7 90 02 e6 f8   6f
c1bc : 20 fd ae 20 9e b7 ca 8a   72
c1c4 : a8 60 20 fd ae 20 9e b7   3a
c1cc : ca 86 d6 20 fd ae 20 9e   a6
c1d4 : b7 ca 86 d3 20 10 e5 60   e7
c1dc : 20 fd ae 20 9e b7 ca ca   13
c1e4 : 8e aa 02 20 fd ae 20 9e   5f
c1ec : b7 ca 8e ab 02 20 ff e9   16
c1f4 : ca ec aa 02 d0 f7 60 fd   69
c1fc : 02 00 ff dd fd ff 00 02   9e
c204 : fd ff 04 02 fd ff 00 02   26
c20c : fd ff 00 02 fd df 00 02   2c
c214 : fd ff 00 02 fd ff 00 02   35
c21c : fd ff 00 02 fd ff 00 02   3d
```

Listing 4. »screen-tool« hilft bei der Bildschirmgestaltung

Spriteset (Listing 5)

Format: SYS 49700,Sprite-Nr,Block-Nr,X-Pos,Y-Pos,Farbe,
Expand (jn)

Spriteset setzt die Parameter für ein Sprite.

SYS 49700,0,11,100,50,6,1 setzt für Sprite Nr. 0 den Block 11 (ab 704) als Datenspeicher fest, positioniert ihn auf die Koordinaten 100/50, färbt ihn blau und vergrößert ihn in X- und Y-Richtung.

```
Name : spriteset               c224 c2ec
-----------------------------------------
c224 : 20 fd ae 20 9e b7 8e b2  3a
c22c : 02 20 fd ae 20 9e b7 8e  86
c234 : b3 02 20 fd ae 20 9e b7  86
c23c : 8e b4 02 20 fd ae 20 9e  bc
c244 : b7 8e b5 02 20 fd ae 20  dd
c24c : 9e b7 8e b6 02 20 fd ae  b7
c254 : 20 9e b7 8e b7 02 ac b2  27
c25c : 02 ad b3 02 99 f8 cf ad  5e
c264 : b2 02 0a a8 ad b4 02 99  6b
c26c : 00 d0 c8 ad b5 02 99 00  8e
c274 : d0 ac b2 02 ad b6 02 99  53
c27c : 27 d0 a2 01 8e b8 02 ae  e8
c284 : b2 02 e8 ca f0 06 0e b8  b4
c28c : 02 4c 87 c2 ae b7 02 e0  61
c294 : 01 f0 1c a9 ff 38 ed b8  34
c29c : 02 8d b8 02 ad 17 d0 2d  04
c2a4 : b8 02 8d 17 d0 ad 1d d0  34
c2ac : 2d b8 02 8d 1d d0 60 ad  9d
c2b4 : 17 d0 0d b8 02 8d 17 d0  18
c2bc : ad 17 d0 0d b8 02 8d 17  cb
c2c4 : d0 ad 1d d0 0d b8 02 8d  86
c2cc : 1d d0 60 00 ff dd 02 00  60
c2d4 : ff fd 02 00 ff dd 02 00  49
c2dc : ff fd 02 00 ff fd 02 00  52
c2e4 : ff fd 02 00 ff fd 02 00  5a
```

Listing 5. »spriteset« erleichtert das Setzen eines Sprites

General Input (Listing 6)

Daß das INPUT-Statement des C64-Basic sehr unkomfortabel ist, darf als altbekannte Tatsache angesehen werden. Man denke nur an die Reaktion des Computers, wenn man Kommata oder Doppelpunkte eingibt. Außerdem kann das Eingabefeld beliebig verlassen, die Maske zerstört werden.

Diese Routine schafft Abhilfe.

Format: SYS 50500,Maximallänge (1 bis 79)

Der Text steht nachher im als erstes definierten String. In der ersten Programmzeile sollte also immer ein Leerstring definiert werden, etwa: 0 IN$=""

Welche Zeichen akzeptiert werden, hängt von der Bitstellung im Kontrollregister (189) ab.

Bit 0 (Wert 1) : Buchstaben A bis Z sind erlaubt
Bit 1 (Wert 2) : Ziffern 0 bis 9 werden akzeptiert
Bit 2 (Wert 4) : Grafikzeichen dürfen eingegeben werden
Bit 3 (Wert 8) : Interpunktionszeichen (.,;:+-*/ etc.)
Bit 4 (Wert 16): kontrolliert, ob während der Eingabe der Cursor blinkt

<DEL> löscht das letzte Zeichen, <HOME> die ganze Eingabe. Dabei wird ein Maskenteil, der sich eventuell rechts vom Eingabefeld befindet, nicht – wie zu erwarten wäre – nach links verschoben, da nicht CHR$(20), sondern CHR$(32)CHR$(157)CHR$(157)CHR$(32)CHR$(157) benutzt wird.

Da der String nicht im Stringspeicher abgelegt wird, sondern direkt nach dem Programm, sollte man ihn nur auslesen, aber hier nicht MID$, RIGHT$ oder LEFT$ benutzen. Sollten diese Funktionen dennoch nötig sein, empfiehlt es sich, einen neuen String anzulegen, der dem alten gleich ist, aber aus Teilen des alten zusammengesetzt wird.

Beispiel: X$=LEFT$(IN$,1)+MID$(IN$,2)

Menü (Listing 7)

Format: SYS 50000,"PUNKT1,PUNKT2,..."

Der Bildschirm zeigt ein Menü mit den Menüpunkten, die im Parameterstring – durch Kommata getrennt – angegeben wurden. Rechts vom ersten Punkt steht ein Pfeil, der mit dem Joystick (Port #2) bewegt werden kann. Ein Druck auf den Feuerknopf übernimmt einen Menüpunkt.

Dessen Nummer steht nachher in 702.

SYS 50000,"EINGABE,AUSGABE,DISKETTE,DIENST, ENDE" wäre zum Beispiel für eine Dateiverwaltung zu gebrauchen.

```
Name : general input      c544 c6ae
-----------------------------------------
c544 : 4c 7b c5 ac ef 07 f0 57  fe
c54c : ce ef 07 20 65 c5 4c a3  d5
c554 : c5 ac ef 07 f0 49 20 65  f1
c55c : c5 ce ef 07 d0 f8 4c a3  b2
c564 : c5 a6 cc a9 01 85 cc a9  a7
c56c : 75 a0 c5 20 1e ab 86 cc  9a
c574 : 60 20 9d 9d 20 9d 00 a5  3a
c57c : bd 29 10 f0 04 a9 00 85  89
c584 : cc a2 00 8e ef 07 8e ee  c2
c58c : 07 20 fd ae 20 9e b7 8e  eb
c594 : ee 07 e0 01 90 04 e0 50  ab
c59c : 90 05 a2 0e 20 37 a4 a6  b5
c5a4 : c6 f0 fc ae 77 02 a0 00  01
c5ac : 84 c6 8c 77 02 e0 14 f0  ff
c5b4 : 92 e0 13 f0 9c e0 20 f0  cc
c5bc : 57 e0 0d d0 03 4c 2f c6  be
c5c4 : a5 bd 29 01 f0 0b e0 41  20
c5cc : 90 07 e0 5b b0 03 4c 14  00
c5d4 : c6 a5 bd 29 02 f0 0b e0  97
c5dc : 30 90 07 e0 3a b0 03 4c  00
c5e4 : 14 c6 a5 bd 29 04 f0 0f  11
c5ec : e0 a1 b0 24 e0 60 90 07  af
c5f4 : e0 80 b0 03 4c 14 c6 a5  6d
c5fc : bd 29 08 f0 10 e0 21 90  1c
c604 : 0c e0 41 b0 08 e0 30 90  50
c60c : 07 e0 3a b0 03 4c a3 c5  d5
c614 : ac ef 07 cc ee 07 f0 87  0d
c61c : 8a 20 d2 ff ee ef 07 ac  4f
c624 : ef 07 99 5d c6 4c a3 c5  92
c62c : 4c a3 c5 a0 02 ad ef 07  2b
c634 : 91 2d a9 00 8d ef 07 c8  cc
c63c : a9 5e 91 2d c8 a9 c6 91  36
c644 : 2d c8 a9 00 91 2d c8 a9  39
c64c : 00 91 2d a9 01 85 cc a9  58
c654 : 00 85 cf a9 20 a4 d3 91  d9
c65c : d1 60 20 20 20 20 20 20  2d
c664 : 20 20 20 20 20 20 20 20  64
c66c : 20 20 20 20 20 20 20 20  6c
c674 : 20 20 20 20 20 20 20 20  74
c67c : 20 20 20 20 20 20 20 20  7c
c684 : 20 20 20 20 20 20 20 20  84
c68c : 20 20 20 20 20 20 20 20  8c
c694 : 20 20 20 20 20 20 20 20  94
c69c : 20 20 20 20 20 20 20 20  9c
c6a4 : 20 20 20 20 20 20 20 20  a4
c6ac : 20 20 20 20 20 20 20 20  ac
```

Listing 6. »general input«, eine schnelle und komfortable INPUT-Routine

```
Name : menue                    c350 c43d
-----------------------------------------
c350 : 20 fd ae 20 57 e2 8d bc  5b
c358 : 02 a6 22 86 9e a6 23 86  bf
c360 : 9f a9 20 20 d2 ff 20 d2  33
c368 : ff a0 01 8c bd 02 88 b1  fb
c370 : 9e c9 2c d0 17 ee bd 02  fc
c378 : 8c bf 02 a9 38 a0 c4 20  75
c380 : 1e ab ac bf 02 c8 ce bc  b2
c388 : 02 4c 6f c3 c8 20 d2 ff  dd
c390 : ce bc 02 d0 da a6 d1 86  8e
c398 : fb a6 d2 86 fc a5 fb 38  c9
c3a0 : e9 50 85 fb d0 02 c6 fc  c4
c3a8 : a6 fb 86 f9 a6 fc 86 fa  8f
c3b0 : ae bd 02 a5 f9 38 e9 50  1c
c3b8 : 85 f9 b0 02 c6 fa ca e0  d7
c3c0 : 02 d0 f0 a6 f9 86 f7 a6  3c
c3c8 : fa 86 f8 a9 01 8d be 02  f4
c3d0 : a0 00 a9 3e 91 f7 20 2b  52
c3d8 : c4 ad 00 dc c9 7e d0 1e  1e
c3e0 : ad be 02 c9 02 90 17 a9  fa
c3e8 : 20 a0 00 91 f7 ce be 02  7f
c3f0 : a5 f7 38 e9 50 85 f7 b0  4f
c3f8 : 02 c6 f8 4c d0 c3 ad 00  07
c400 : dc c9 7d d0 1f ad be 02  99
c408 : cd bd 02 b0 17 a9 20 a0  cb
c410 : 00 91 f7 ee be 02 a5 f7  37
c418 : 18 69 50 85 f7 90 02 e6  83
c420 : f8 4c d0 c3 ad 00 dc c9  cd
c428 : 6f d0 ae a0 ff 88 d0 fd  42
c430 : ad 00 dc c9 7f d0 f9 60  75
c438 : 0d 0d 20 20 00 c5 a9 08  bd
```

Listing 7. »menue«, Auswahlmenü leichtgemacht

Disk-Tool (Listing 8)

Disk-Tool umfaßt einige Routinen, die den Umgang mit der Floppy erleichtern.

1) Programm absolut speichern
   SYS 50250,"NAME",Startadresse,Endadresse
2) Programm absolut laden
   SYS 50253,"NAME"Startadresse
3) Disk-Kommando senden
   SYS 50256,"Kommando"
   Anschließend wird der Diskstatus ausgelesen
4) Disk-Status auslesen
   SYS 50256,""

```
Name : disk-tool           c44a c4ec
------------------------------------
c44a : 4c 53 c4 4c 91 c4 4c b9  de
c452 : c4 20 fd ae 20 57 e2 a6  11
c45a : 22 a4 23 20 bd ff a9 31  80
c462 : a2 08 a0 01 20 ba ff 20  69
c46a : c0 ff 20 fd ae 20 8a ad  63
c472 : 20 f7 b7 a5 14 85 c1 a5  f0
c47a : 15 85 c2 20 fd ae 20 8a  f1
c482 : ad 20 f7 b7 a5 14 85 ae  a3
c48a : a5 15 85 af 4c ed f5 20  5d
c492 : fd ae 20 57 e2 a6 22 a4  0f
c49a : 23 20 bd ff a9 31 a2 08  fb
c4a2 : a0 01 20 ba ff 20 c0 ff  26
c4aa : 20 8a ad 20 f7 b7 a6 14  7f
c4b2 : a4 15 a9 00 4c 9e f4 20  19
c4ba : fd ae 20 57 e2 a6 22 a4  37
c4c2 : 23 20 bd ff a9 31 a2 08  23
c4ca : a0 0f 20 ba ff 20 c0 ff  55
c4d2 : a2 31 20 c6 ff 20 cf ff  2e
c4da : 20 d2 ff a5 90 29 40 f0  4d
c4e2 : f4 20 cc ff a9 31 20 c3  45
c4ea : ff 60 20 20 d2 ff 4c e5  50
```

Listing 8. »disk-tool« erleichtert die Arbeit mit der Floppy

Programmiertips

Soll eine Variable ihren Wert in Abhängigkeit von einer anderen erhalten, ohne daß eine Proportion besteht, benötigt man für jede Möglichkeit eine Zeile, zum Beispiel:

```
10 IFA= 5THENB=3261
20 IFA= 8THENB=7901
30 IFA= 9THENB=2079
40 IFA=17THENB= 681
50 IFA=99THENB=   3
```

Durch clevere Anwendung des Vergleiches (»PRINT A=5« liefert 0 wenn A nicht 5 ist, beziehungsweise -1, wenn A gleich 5 ist) kann man auch dieses kleine Programm auf eine Zeile verkürzen:

```
10 B=0-3261*(A=5)-7901*(A=8)-2079*(A=9)-681*(A=17)-3*(A=99)
```

Diese Methode spart Zeit, Speicherplatz und Tipparbeit und erhöht darüber hinaus die Übersichtlichkeit des Listings, da der Lesende durch die fünf Zeilen nicht aus dem Zusammenhang gerissen wird.

Oft will man den Wert einer Variablen abhängig vom alten Wert ändern. Zum Beispiel: Wenn X vorher gleich 8 war, soll es jetzt 5 werden und umgekehrt.

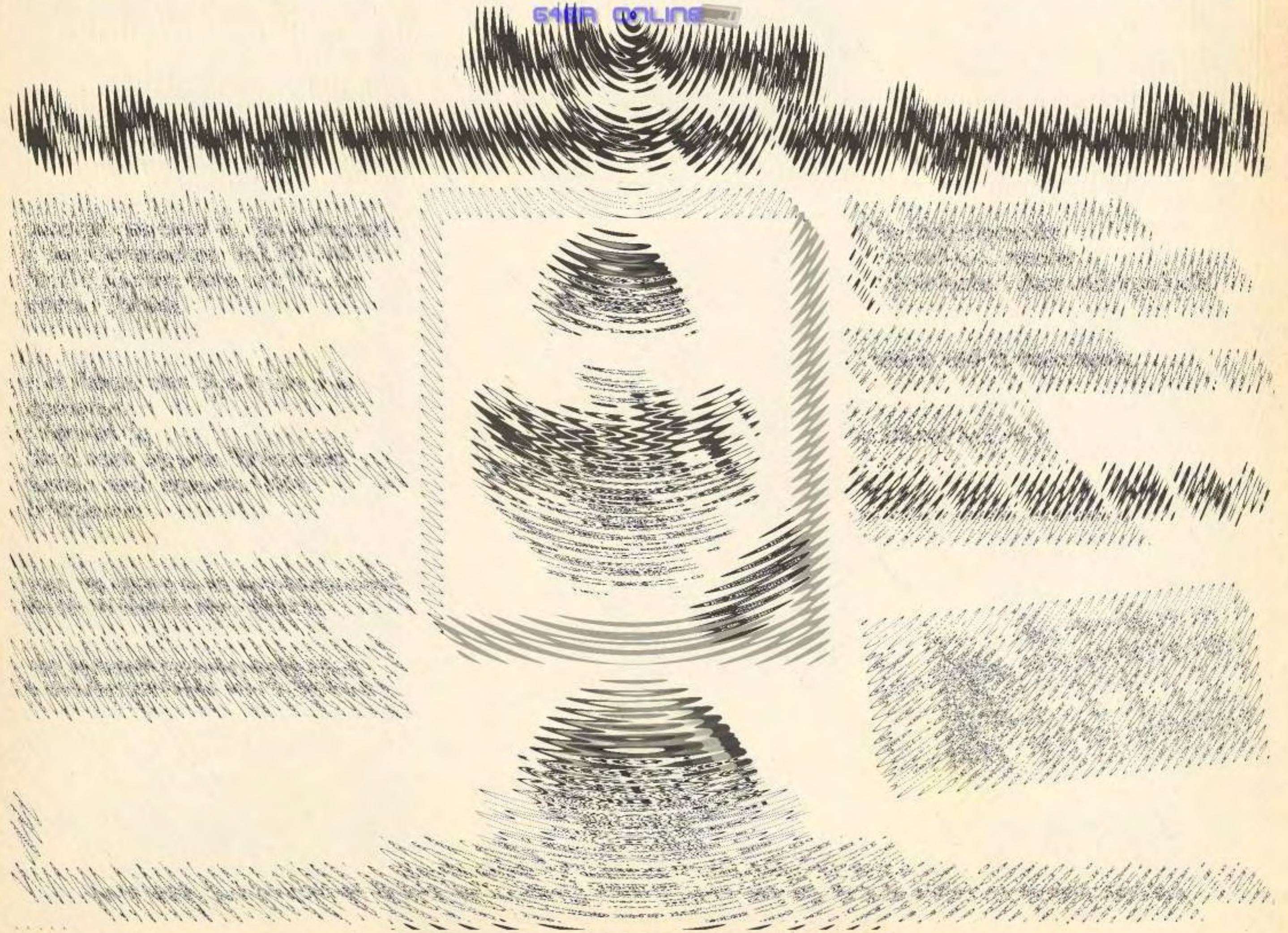
Normalerweise benötigt der Programmierer zwei Zeilen:

```
10 IFX=8THENX=5:GOTO30
20 IFX=5THENX=8
```

Wendet man die ABS-Funktion geschickt an, reicht eine Zeile schon aus:

```
10 IFX=5 OR X=8THENX=ABS(X-8)+5
```

(Stefan Meierhofer/tr)



Impressum

Herausgeber: Carl-Franz von Quadt, Otmar Weber

Chefredakteur: Michael Scharfenberger

Stellv. Chefredakteur: Albert Absmeier

Leitender Redakteur: Georg Klinge

Redaktion: Herbert Buckel (bj), Roland Fieger (rf), Achim Hübner (ah), Norbert Jungmann (nj), Dieter Mayer (dm), Jörg Kähler (jk), Markus Ohnesorg (og), Thomas Röder (tr), Karsten Schramm (ks)

Titelfoto: Jens Jancke

Titelgestaltung: Heinz Rauner Grafik-Design

Layout:
Leo Eder (Ltg.), Sigrid Kowalewski (Cheflayouterin), Rolf Raß, Katja Milles

Produktionsleiter: Klaus Buck

Anzeigenverkaufsleitung: Ralph-Peter Rauchfuss

Anzeigenverkauf: Helmut Distl (398)

Auslandsrepräsentation:

Schweiz: Markt&Technik Vertriebs AG, Kollerstr. 3, CH-6300 Zug, Tel. 042-41 56 56, Telex: 862 329

USA: M&T Publishing Inc.; 501 Galveston Drive Redwood City, CA 94063 Telefon: (415) 366-3600

Manuskripteinsendungen: Manuskripte und Programmlistings werden gerne von der Redaktion angenommen. Sie müssen frei sein von Rechten Dritter. Sollten sie auch an anderer Stelle zur Veröffentlichung oder gewerblichen Nutzung angeboten werden, so muß dies angegeben werden. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck in von der Markt&Technik Verlag AG herausgegebenen Publikationen und zur Vervielfältigung der Programmlistings auf Datenträger. Mit der Einsendung von Bauanleitungen gibt der Einsender die Zustimmung zum Abdruck in von Markt&Technik Verlag AG verlegten Publikationen und dazu, daß Markt&Technik Verlag AG Geräte und Bauteile nach der Bauanleitung herstellen läßt und vertreibt oder durch Dritte vertreiben läßt. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen.

Marketingleiter: Hans Hörl (114)

Vertriebsleiter: Helmut Grünfeldt (189)

Anzeigenverwaltung und Disposition: Michaela Hörl

Verlagsleiter M&T-Buchverlag: Günther Frank (212)

Druck: SOV St. Otto-Verlag GmbH, Laubanger 23, 8600 Bamberg

Bezugsmöglichkeiten: Leser-Service: Telefon (089) 4613-249. Bestellungen nimmt der Verlag oder jede Buchhandlung entgegen.

Preis: Das Einzelheft kostet DM 14,–

Vertrieb Handelsauflage: Inland (Groß-, Einzel- und Bahnhofsbuchhandel) sowie Österreich und Schweiz: Pegasus Buch- und Zeitschriften-Vertriebs GmbH, Hauptstätter Straße 96, 7000 Stuttgart 1, Telefon (0711) 6483-0

Verantwortlich:
Für redaktionellen Teil: Michael Scharfenberger
Für Anzeigen: Britta Fiebig

Redaktionsdirektor: Michael M. Pauly

Vorstand: Carl-Franz von Quadt, Otmar Weber

Anschrift für Verlag, Redaktion, Vertrieb, Anzeigenverwaltung und alle Verantwortlichen:
Markt & Technik Verlag Aktiengesellschaft, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0, Telex 5-22052



